TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

JARKKO KOISTINAHO

KORKEAN SAATAVUUDEN JÄRJESTELMÄN AUTOMAATTINEN TESTAUS

Diplomityö

Tarkastaja: Mika Katara
Tarkastaja ja aihe hyväksytty
Tieto- ja sähkötekniikan tiedekunta-
neuvoston kokouksessa 03.04.2013

TIIVISTELMÄ

TAMPEREEN TEKNILLINEN YLIOPISTO
Tietotekniikan koulutusohjelma
KOISTINAHO, JARKKO: Korkean saatavuuden järjestelmän automaattinen testaus
Diplomityö, 58 sivua, 17 liitesivua
Kesäkuu 2013
Pääaine: Ohjelmistotuotanto
Tarkastaja: Mika Katara
Avainsanat: automaattinen testaus, korkea saatavuus, suorituskykytestaus, kestotestaus, mallipohjainen testaus

Saatavuus on tärkeä osa kriittisiä tietojärjestelmiä ja heikon saatavuuden omaavat järjestelmät voivat toimintahäiriöiden seurauksena aiheuttaa suuria taloudellisia tappioita tai jopa hengenvaaraa. Tämän työn tutkimusongelma on, miten saatavuutta voidaan mitata. Tutkimusongelmaa yritetään ratkaista automaattisella testauksella, jonka avulla pyritään mittaamaan testattavan järjestelmän saatavuutta.

Automaattisella testauksella yritetään tuottaa kuormaa pitkäkestoisesti testattavaan järjestelmään ja tutkia, miten järjestelmä käyttäytyy. Manuaalinen testaus ei ole tarpeeksi riittävä kuorman tuottamiseen vakaasti ja luotettavasti järkevien resurssien puitteissa.

Tässä työssä pyritään mittaamaan saatavuutta useiden eri testausmenetelmien avulla. Saatavuuden testaamisessa sovelletaan toiminnallista testausta, haavoittuvuus-, suorituskyky-, harmaalaatikkotestausta ja mallipohjaista testausta. Saatavuutta mitataan käyttäjän näkökulmasta.

Saatavuuden mittaamista varten kehitettiin testipeti. Sen avulla voidaan tutkia potentiaalisia pullonkauloja ja löytää virheitä testattavasta järjestelmästä mittauksen lisäksi. Testauskohteina olivat uusi kehitettävä hätäkeskustietojärjestelmä ERICA ja sen osana toimiva Varotietopalvelu.

Tässä työssä pääpaino oli testipedin kehityksessä sekä siinä, että voidaanko automaattisen testauksen avulla mitata korkeaa saatavuutta. Testaukseen valikoitiin vain oleellisimmat järjestelmän toiminnot, joita vasten suoritettiin pitkäkestoisia testejä. Testituloksia on arvioitava huolellisesti, koska testaukset suoritettiin tietyllä aikavälillä ja mittaus on arvio, pääseekö järjestelmä palvelutasosopimuksissa asetettuihin lupauksiin. Todellisuudessa korkean saatavuuden järjestelmien täytyy olla toimintakykyisiä kuukausien ajan ja testauksella voidaan vain saada arvioita järjestelmän toimintakyvystä.

ABSTRACT

TAMPERE UNIVERSITY OF TECHNOLOGY
Master's Degree Programme in Information Technology
KOISTINAHO, JARKKO: Automated Testing of a High Availability System
Master of Science Thesis, 58 pages, 17 appendix pages
June 2013
Major: Software engineering
Examiner: Mika Katara
Keywords: automated testing, high availability, performance testing, endurance testing, model-based testing

Availability is a crucial part of critical systems and poor availability in the worst case can result in major economic loss or can threaten life. Topic of this thesis is how high availability can be measured. The study aims to measuring availability by automated testing.

Automatic testing targets to produce long-term load to system under test and tries to examine how the system behaves. Manual testing is not sufficient for producing stable and reliable long-term load.

This thesis aims at measuring the high availability by functional, fuzz, performance, gray-box and model-based testing. The high availability is measured by the user's point of view.

High Availability Testbed was developed for measuring high availability of system under test. System errors, bugs and potential bottlenecks are possible to be discovered and explored with testbed. Testbed was developed for measuring the high availability of a new Finnish emergency response system called ERICA and Caution Information System as part of its. Both systems are currently under development.

In this thesis the focus was in developing the testbed and could high availability be measured by automated testing. Only the most essential functions of the system were selected for long-term testing. Test results should be carefully evaluated because the tests were performed in a specific time frame and the measurement is an estimate, could the system in test fulfill the promises of service level agreement. In reality, high-availability systems need to be functional for months, and testing can only measure estimates of system's ability to function.

ALKUSANAT

Tämä diplomityö on tehty Insta DefSec Oy:n hätäkeskustietojärjestelmän kehitysprojektissa esitetystä aiheesta. Opin paljon uusia asioita sekä aiheesta että ohjelmistokehityksestä, vaikka työn aihe rajautuu pieneen osaan kokonaisuudesta.

Aluksi haluan kiittää kaikkia diplomityöhön vaikuttaneita osapuolia sekä testipedin kehittämiseen osallistuneita työtovereitani. Haluan kiittää työn ohjaamisesta ja tarkastamisesta Mika Kataraa sekä aiheen ehdottamisesta, hyvistä neuvoista, työn ohjaamisesta ja kommentoinnista Aapo Koskea. Lopuksi esitän kiitokseni sekä perheelleni että ystävilleni tuesta diplomityön kirjoittamisen ja opiskelun aikana.

Tampereella 13.05.2013

Jarkko Koistinaho

SISÄLLYS

TERMIT JA NIIDEN MÄÄRITELMÄT

HA	Lyhenne termistä High Availability eli korkea saatavuus.
HA Core	Testejä suorittava komponentti testipedissä. Sisältää testauslogiikan.
HA Miniclient	Testipedin testisovellus, joka voi simuloida asiakassovellusta tai kietoa sen toteutuksen kokonaan testien käyttöön.
HA Test Agent	Asiakaskoneella oleva sovellus, jonka kautta viestitetään Miniclientteja käyntiin.
HA Toolbox	Sovellus, jonka avulla voidaan tutkia testien tuloksia ja ohjata Miniclienttien käynnistystä tai sammutusta.
Haavoittuvuustestaus	Testausmenetelmä, jossa pyritään häiritsemään testattavaa järjestelmää virheellisillä syötteillä tai satunnaisuudella. (engl. fuzz testing) [1, s. 22]
Harmaalaatikkotestaus	Testausmenetelmä, joka pyrkii hyödyntämään sekä mustalaatikko- että lasilaatikkotestauksen hyvät puolet. (gray-box testing) [2, s. 207-209]
Kestotestaus	Kestotestauksessa ajetaan testejä jatkuvalla syötöllä ja järjestelmää kuormitetaan vakiokuormalla riittävän pitkällä aikavälillä. (endurance/soak testing) [3, s. 39]
Korkea saatavuus	Eräs tietojärjestelmien suunnittelun käytäntö, jonka tarkoituksena on pyrkiä takaamaan, että järjestelmä on käyttäjien saatavissa silloin, kun sen käytölle on oikeasti tarvetta. (high availability) [4]
Kuormitustestaus	Kuormitustestauksessa ajetaan testejä tietyllä aikavälillä ja kuormalla. (load testing) [3, s. 39]
Lasilaatikkotestaus	Testausmenetelmä, missä testaaja tutkii testattavan sovelluksen koodia ja pyrkii tekemään sen pohjalta suoritettavat testitapaukset. (white-box testing) [2, s. 56]
Luotettavuus	Laatutekijä, joka varmentaa sen, että järjestelmän on kyettävä täyttämään sille asetetut vaatimukset eri tilanteissa tietyn ajan kuluessa. (reliability) [5, s. 199]

Mallipohjainen testaus	Testausmenetelmä, jossa muodostetaan järjestelmän käyttäytymistä kuvaavia malleja, joiden perusteella mallipohjainen testaustyökalu generoi testit. (model-based testing) [6, s. 6-7]
Mustalaatikkotestaus	Testausmenetelmä, jossa järjestelmää testataan syötteiden ja vasteiden avulla, kun lähdekoodia ei ole välttämättä saatavilla tai kun sitä ei haluta hyödyntää testauksessa. (black-box testing) [2, s. 56-57]
Operationaaliset profiilit	Kvantitatiivinen jäsennelmä, joka kuvaa miten järjestelmää käytetään. (operational profile) [7, s. 159]
Rasitustestaus	Rasitustestauksessa pyritään hakemaan raja-arvoja eri komponenteille kuormittamalla järjestelmää mahdollisimman suurella kuormalla. (stress testing) [3, s. 39]
Saatavuus	Laatutekijä, jolla pyritään varmentamaan, että järjestelmä on käytettävissä silloin, kun sitä tarvitaan. (availability) [5, s. 194-195]
Skaalautuvuus	Laatutekijä, joka tarkoittaa järjestelmän toimintakyvyn säätelyä tarpeen mukaan häiriöttä. (scalability) [5, s. 200-201]
Suorituskykytestaus	Suorituskykytestauksessa pyritään selvittämään järjestelmän kykyä selviytyä tehtävistään vaaditussa ajassa erilaisissa tilanteissa. (performance testing) [3, s. 39]
Tehokkuus	Laatutekijä, jolla tarkoitetaan järjestelmien tai niiden toimintojen kykyjä käsitellä paljon kuormaa tietyssä ajassa. (efficiency/performance) [5, s. 198-199]
Testijoukko	Ryhmä testitapauksia. (test suite)
Testipeti	Korkean saatavuuden testausta varten kehitetty testaustyökalu. (high availability testbed)
Testitapaus	Yksittäinen testi. (test case)
Toiminnallinen testaus	Mustalaatikkotestauksen muoto, jonka tarkoituksena on testata järjestelmälle asetettuja vaatimuksia. (functional testing) [2, s. 57]

Ylläpidettävyys	Laatutekijä, jolla pyritään mittaamaan toimitetun järjestelmän ylläpitoa, joka kattaa virheiden korjauksen, tarpeellisten muutosten teon ja uusien ominaisuuksien lisäysmahdollisuuden. (serviceability/maintainability) [5, s. 196-197]

1. JOHDANTO

Ohjelmistojen testaus on merkittävä osa laadun varmistusta. Sen tarkoituksena on varmistaa, että testattava järjestelmä toimii asetettujen tai oletettujen vaatimusten mukaisesti määritellyissä ympäristöissä. Testauksella on kaksi päätavoitetta: mitata testattavan kohteen laatua ja testata kohdetta määrittelyn asettamien vaatimusten suhteen. Yleinen harhaanjohtava kuva testauksesta on se, että testaus on vain ohjelmistossa olevien virheiden etsintää ja niiden analysointia. Edellä mainittujen lisäksi testaajan tulee huomioida testattavan kohteen määrittelyn ja vaatimusten tuomat rajat. Järjestelmän tulee olla toimintakuntoinen ja testauksessa havaitut merkittävät viat täytyy olla korjattuna ennen järjestelmän toimitusta asiakkaalle.

Korkea saatavuus on suunnittelukäytäntö, millä pyritään varmistamaan, että järjestelmä on poikkeuksetta käyttäjien käytettävissä silloin, kun käytölle on oikeasti tarvetta. Saatavuus on tärkeä laatutekijä kriittisissä järjestelmissä, jotka voivat toimintahäiriön seurauksena aiheuttaa suuria taloudellisia tappioita tai johtaa jopa ihmishenkien menetyksiin. Kriittisiksi järjestelmiksi luokitellaan kaikki sellaiset järjestelmät, jotka voisivat vikaantuessaan aiheuttaa vaaraa ympäristölleen.

Automaattinen testaus pyrkii täydentämään manuaalisen testauksen aukkoja, mutta samalla se voi lisätä tai vähentää testaajan työtä. Ohjelmistoprojekteissa on varattu enemmän tai vähemmän resursseja testaukseen, jolloin on tärkeää, että testaaja voi keskittyä oleellisiin tehtäviin. Automaattisella testauksella voidaan esimerkiksi mitata järjestelmän tehokkuutta, johon manuaalinen testaus ei kykenisi, vaikka resursseja olisi käytettävissä äärettömästi. Automaattinen testaus voi säästää aikaa, rahaa ja mahdollisesti parantaa testauksen laatua. Joitakin testejä, kuten esimerkiksi kuormittavia testejä, ei ole mahdollista suorittaa manuaalisen testauksen voimin järkevällä työpanoksella.

Oikean testaustyökalun valinnassa on tärkeää huomioida testattavan kohteen sekä testaajien tai muun henkilöstön asettamat vaatimukset. Testaustyökalun valinnassa tulee huomioida myös henkilöstön osaaminen sekä käytettävät testausmenetelmät ja mittauspisteet. Väärän testaustyökalun valinta vaikuttaa suuresti ajan ja muiden resurssien käyttöön. Testaustyökalun valintaan voi vaikuttaa testaajien vaatimukset testaustyökalun ominaisuuksilta eikä se välttämättä täytä kaikkia projektin testaustarpeita tai vaatimuksia.

Tämän työn tutkimusongelma on, miten kriittisen järjestelmän korkeaa saata-

vuutta voidaan mitata. Tutkimusongelmaa yritetään ratkaista automaattisella testauksella, jonka avulla pyritään mittaamaan ja tutkimaan testattavaa järjestelmää. Mittaus tehdään korkean saatavuuteen liittyvien laatutekijöiden näkökulmasta. Saatavuuden lisäksi mitattavat laatutekijät ovat luotettavuus, tehokkuus, skaalautuvuus ja ylläpidettävyys.

Yleensä järjestelmät koostuvat monista eri komponenteista tai palveluista, jolloin saatavuutta ei voida mitata vain yksittäisten palveluiden osalta. Korkea saatavuus on otettava kokonaisuutena huomioon.

Tässä työssä keskitytään käyttäjäkokemukseen perustuvaan korkeaan saatavuuteen, koska käyttäjät ovat kehitettävien järjestelmien käytön kannalta tärkein sidosryhmä. Järjestelmän vaivaton käyttö on eräs käyttäjien vaatima ominaisuus, mutta käyttäjien ei tarvitse olla tietoisia siitä, mitä järjestelmässä tapahtuu, jos sen kriittiset palvelut ovat kykeneviä toimimaan niin kuin pitäisi. Tällöin hyvä lähtökohta on käyttäjän kokema saatavuus ja sen kautta mitattava tieto, kuinka saatava järjestelmä on.

Tässä työssä ei ole kuitenkaan tarkoitus käydä kaikkia testattavan järjestelmän toimintoja läpi ja testata niitä, vaan keskitytään vain keskeisiin toimintoihin. Kuormaa aiheuttavien testien avulla voidaan pyrkiä mittaamaan saatavuutta.

Työssä korkeaa saatavuutta mitataan useiden eri testausmenetelmien avulla. Tässä hyödynnetään toiminnallista testausta, haavoittuvuus-, suorituskyky-, harmaalaatikkotestausta ja mallipohjaista testausta. Mittausta tehdään oikeellisten syötteiden avulla ja niiden lisäksi haavoittuvuustestausta suoritetaan satunnaisuuden perusteella, koska järjestelmän käyttäjän tekemät päätökset ja toimet eivät ole ennakoitavissa. Tyypillisesti haavoittuvuustestauksessa syötetään virheellisiä syötteitä ja tutkitaan toimiiko järjestelmä toivotulla tavalla.

Korkean saatavuuden testaamiseksi on otettava huomioon kuormittavia testejä. Kuormittavia testausmenetelmiä ovat kesto-, rasitus-, kuormitus-, ja volyymitestaus. Tässä työssä on jätetty testauksen ulkopuolelle volyymitestaus, koska se ei sovellu korkean saatavuuden mittaukseen ja koska rasitustestauksessa pyritään mittaamaan osittain myös volyymia. Rasitus- ja kuormitustestauksen hyödyntämiseksi korkean saatavuuden mittaukseen tutkitaan teorian tasolla, koska niitä voidaan käyttää vasta silloin, kun testattavan järjestelmän kehitys ja toiminnallisuus on stabiloitunut selvästi.

Korkean saatavuuden mittaamiseksi kehitettiin testipeti, jonka avulla voidaan havaita ja tutkia potentiaalisia ongelmia esimerkiksi luotettavuuden, tehokkuuden ja saatavuuden suhteen. Testipeti tehtiin uuden kehitettävän hätäkeskustietojärjestelmän testausta varten ja pyrkimyksenä on saada tarkkaa tietoa kyseisen järjestelmän tilasta korkean saatavuuden osalta.

Työn rakenne on seuraava: luvussa 2 on työn teoriaosuus, jossa kerrotaan työn

taustoista, käytettävistä testausmenetelmistä sekä kuvataan testattavia järjestelmiä. Testipedin rakenteesta sekä käytetyistä testaustekniikoista, menetelmistä, ja -aineistoista kerrotaan luvussa 3. Luvussa 4 kuvataan testipedin kehitysvaihe sekä esitetään työn tulokset ja jatkokehitys. Jatkokehityksessä pohditaan jatkokehitysajatusten vaikutusta korkean saatavuuden mittaukseen. Luvussa 5 esitetään loppupäätelmät.

2. TYÖN TAUSTAT

Tietoteknisiä järjestelmiä testataan testauksen V-mallin mukaisesti neljällä eri testitasolla: yksikkö-, integraatio-, järjestelmä-, ja hyväksyntätestaus. Yksikkötestauksessa keskitytään testaamaan vain yhtä metodia, luokkaa tai komponenttia, jota kehittäjä testaa sen valmistuessa. Integraatiotestauksessa yhdistetään useita komponentteja moduuleiksi, jolloin testataan komponenttien välinen kommunikointi ja toimivuus yhdessä. Järjestelmätestauksessa moduulit yhdistetään kokonaisuudeksi ja testataan niitä. Viimeinen testausvaihe on hyväksyntätestaus, minkä tarkoituksena on varmentaa järjestelmän laatu ja toimivuus ennen sen luovuttamista asiakkaalle.

Kriittisten järjestelmien suunnittelussa otetaan huomioon, kuinka saatavissa järjestelmä on sen käyttäjille. Tämän työn tutkimusongelma on, miten kriittisen järjestelmän korkeaa saatavuutta voitaisiin mitata. Saatavuutta ei voida mitata suoraan yksikkö- ja integraatiotestauksessa, koska kriittiset järjestelmät koostuvat monista eri osista. Jos järjestelmän saatavuutta mitattaisiin vain yksikkö- tai integraatiotestaustasolla, testitulokset voisivat antaa liian hyvän kuvan järjestelmän saatavuudesta kokonaisuutena.

Silloin olennaiset testitasot korkean saatavuuden mittauksen kannalta ovat järjestelmä- ja hyväksyntätestaus. Pääpaino on kuitenkin järjestelmätestaustasolla, koska hyväksyntätestaus on tarkoitettu toimituksen kynnyksellä tapahtuvaksi manuaaliseksi testaukseksi, jolloin asiakas testaisi järjestelmää haluamallaan tavalla. Hyväksyntätestaus on tyypillisesti rajallinen ajaltaan, jolloin lyhyessä ajassa ei voitaisi testata järjestelmää tarpeeksi kattavasti, tehokkaasti tai luotettavasti saatavuuden kannalta.

2.1 Korkea saatavuus

Kriittisten järjestelmien olennaisin laatutekijä on saatavuus — on erityisen tärkeää, että kriittinen järjestelmä on saatavissa koska tahansa, jos käyttäjä tai toinen järjestelmä on riippuvainen kyseisestä järjestelmästä. Korkea saatavuus on suunnittelukäytäntö, jonka tarkoituksena on pyrkiä takaamaan, että järjestelmä on käyttäjien saatavissa silloin, kun sen käytölle on oikeasti tarvetta. Schmidt määrittelee korkean saatavuuden, että se on järjestelmän ominaisuus, jolla pyritään suojautumaan tai toipumaan pieniltä katkoksilta automatisoidusti [4].

Korkean saatavuuteen liittyviä laatutekijöitä ovat Schmidtin mukaan saatavuus (engl. availability), luotettavuus (engl. reliability) ja ylläpidettävyys (engl. serviceability/ maintainability) [4]. Suurissa ja kriittisissä järjestelmissä tiedon hajauttaminen ja replikointi ovat avainasemissa, jolloin Ladin et al:in mainitsemat tehokkuus (engl. efficiency/performance) ja skaalautuvuus (engl. scalability) on otettava huomioon korkean saatavuuden mittauksessa [8]. Saatavuus on laatutekijä, joka ilmentää sitä, kuinka kauan järjestelmä on tarvittaessa käytettävissä [5, s. 194-195].

Luotettavuus on laatutekijä, joka kattaa saatavuuden, turvallisuuden ja varmuuden järjestelmän toiminnasta. Käsite voidaan lyhentää tarkoittamaan seuraavaa: järjestelmän tulee toimia vaatimuksien mukaisesti eri tilanteissa tietyn ajan kuluessa [5, s. 199]. Luotettavuuteen kuuluu myös virheensietokyky, johon sisältyy poikkeaman havaitseminen, sen eristäminen, korjaaminen ja siitä toipuminen.

Skaalautuvuudella tarkoitetaan järjestelmän toimintakyvyn säätelyä tarpeen mukaan häiriöttä. Skaalautuvuus on tärkeä laatutekijä kriittisissä järjestelmissä sillä niiden tehokkuutta on oltava mahdollista kasvattaa ilman sen toiminnan häiriintymistä, kun järjestelmien toimintavaatimukset kasvavat. Skaalautuvuus luokitellaan kahteen eri tyyppiin: vertikaaliseen ja horisontaaliseen. Vertikaalisessa skaalautumisessa kasvatetaan yhden järjestelmän resursseja tarpeiden mukaiseksi; horisontaalisessa skaalautumisessa monistetaan yksi järjestelmä monen järjestelmän kokonaisuudeksi. [5, s. 200-201]

Laatutekijänä tehokkuus kuuluu osittain skaalautuvuuteen, mutta se on hyvä käsitellä omana kokonaisuutenaan. Skaalautuvuutta ei voida mitata tai arvioida, jos tehokkuutta ei pystytä mittaamaan. Tehokkuudella tarkoitetaan järjestelmän kykyä käsitellä paljon kuormaa tietyssä ajassa [5, s. 198-199]. Esimerkiksi järjestelmän palvelimen tehokkuutta voidaan mitata onnistuneiden kyselyiden määrillä suhteessa aikaan. Tehokkuus ei voi aina tarkoittaa laadukasta tiedon käsittelyä, sillä tehokkuus voi olla palvelun kannalta älykästä, jolloin pyritään palvelemaan käyttäjiä mahdollisimman tehokkaasti heikommalla laadulla.

Ylläpidettävyys on laatutekijä, jonka vaikutus näkyy selvimmin, kun järjestelmä on toimitettu asiakkaalle ja se on jo käytössä. Siihen sisältyy virheiden havaitsemista ja korjaamista, tarpeellisten muutosten toteuttamista, sekä uusien ominaisuuksien lisäämistä vähällä vaivalla [5, s. 196-197]. Ohjelmistokehityksessä voidaan ennakoida tulevia muutostöitä ja parantaa järjestelmän ylläpidettävyyttä ennen kuin se on asiakkaan käytössä. Kriittisissä järjestelmissä ei ole välttämättä varaa jättää ylläpitovaiheeseen esimerkiksi tarpeellisia päivityksiä, jolloin ylläpidettävyyden merkitys voi olla ratkaiseva.

Kriittisten järjestelmien suunnittelussa korkea saatavuus määritellään palvelutasosopimuksissa prosentteina ajasta. Palvelutasosopimus on järjestelmän toimittajan ja asiakkaan välinen sopimus, joka pyrkii varmentamaan toimitettavan järjestelmän

vaatimustasot ja niiden toteutuvuuden asiakkaalle. Tyypillisesti tasojen alittamisesta seuraa sovittu sanktio toimittajalle. Suurin ongelma on siinä, miten voidaan varmistua siitä, että toimitettava järjestelmä on laadukas ja kykenee palvelutasosopimuksen asettamiin vaatimuksiin.

Korkean saatavuuden määrittely ei sisällä suunniteltuja katkoksia, kuten huoltoikkunan aiheuttamaan katkosta. Korkean saatavuuden järjestelmiä mainostetaan yhdeksikköjen määrällä. Esimerkiksi viiden yhdeksikön korkean saatavuuden järjestelmä voi olla määritelty olevan saatavissa 99,9995 % vuoden aikana, jolloin järjestelmällä voi olla katkoksia vuoden aikana enintään 2 minuuttia ja 37 sekuntia.

Korkean saatavuuden määrittämisessä on edellytyksenä, että kaikki palvelut sisältävät vähintään saman lupauksen saatavuuden suhteen kuin koko järjestelmän saatavuus on. Eli jos jollakin järjestelmän palvelulla on saatavuuslupaus 99,96 % niin koko järjestelmän saatavuus on korkeintaan 99,96 %. Tällöin järjestelmän toimittajan on keksittävä jokin muu ratkaisu saatavuuden parantamiseen. Päästäkseen tavoitteeseensa, toimittajan tulee ottaa huomioon saatavuuden ja luotettavuuden lisäksi myös muitakin seikkoja, jotka vaikuttavat korkeaan saatavuuteen.

Yhdeksikköjen määrän kasvaessa ne kadottavat merkityksensä: käytännössä järjestelmä ei voi olla koskaan alhaalla, jos yhdeksikköjä on tarpeeksi. Tällöin järjestelmältä odotetaan täydellistä luotettavuutta ja saatavuutta.

Korkeaan saatavuuteen liittyvät laatutekijät ovat enemmän tai vähemmän sidottuja toisiinsa. Esimerkiksi korkea luotettavuus voi vähentää tehokkuutta, mutta kuitenkin voi lisätä ylläpidettävyyttä, kuten taulukko 2.1 osoittaa. Taulukko esittää aiemmin mainittujen laatutekijöiden vaikutusta toisiinsa, kun järjestelmän laatutekijät muuttuvat parempaan tai huonompaan suuntaan.

Plus-merkillä merkityt taulukon solut merkitsevät sitä, että kyseiset laatutekijät korreloivat toisiaan positiivisesti, ja miinuksella merkityt laatutekijät vaikuttavat päinvastaisesti. Tyhjät solut tarkoittavat, ettei kyseisillä laatutekijöillä ole suoraa

Taulukko 2.1: Laatutekijöiden korreloituminen (mukailtu lähteestä [9]).

	Luotettavuus	**Saatavuus**	**Skaalautuvuus**	**Tehokkuus**	**Ylläpidettävyys**
Luotettavuus		+	+/−	−	+
Saatavuus	+		+		+
Skaalautuvuus	+/−	+		+	−
Tehokkuus	−		+		−
Ylläpidettävyys	+	+	−	−	

vaikutusta toisiinsa. Taulukko on mukailtu Gnanasekaran [9] laatutekijöiden vaikutustaulukosta.

Gnanasekaran taulukossa ei ole skaalautuvuutta laatutekijänä, joten sen vaikutusta muihin laatutekijöihin on sovellettu korkean saatavuuden näkökannalta. Skaalautuvuuden ja luotettavuuden välinen suhde voi aiheuttaa molemmissa laatutekijöissä tietyissä tapauksissa positiivista tai negatiivista korrelointia. Skaalautuvuus voi lisätä järjestelmän monimutkaisuutta, mikä heikentäisi luotettavuutta — kuitenkin tiedon replikointi voi parantaa molempia laatutekijöitä.

Piedad et al. korostavat, että saatavuus on käyttäjäkokemukseen perustuva mittari ja sitä täytyy mitata käyttäjän näkökulmasta alusta loppuun (engl. end-to-end). Käyttäjän mielestä järjestelmä on saatavissa vain silloin, kun järjestelmää on mahdollista käyttää tarvittaessa. He lisäävät, että monet IT-alan yritykset ovat ymmärtäneet saatavuuden merkityksen väärin. Osa yrityksistä uskoo, että vain järjestelmän eri kokonaisuuksien, kuten palvelimen tai verkkoyhteyksien, saatavuudesta tulisi huolehtia; osa yrityksistä pitää yksittäisten kriittisten järjestelmäkomponenttien saatavuuden mittausta riittävänä. [10, s. 19]

Tässä työssä keskitytään käyttäjäkokemukseen perustuvaan korkeaan saatavuuteen, koska käyttäjät ovat kehitettävän järjestelmän tärkein sidosryhmä. Käyttäjille on erittäin tärkeää järjestelmän vaivaton käyttö eikä käyttäjien tarvitse olla tietoisia siitä, mitä järjestelmässä yksityiskohtaisella tasolla tapahtuu. Tämän nojalla kriittisten järjestelmien suunnittelussa saatavuus pyritään takaamaan vain käyttäjälle eikä yksittäisen palvelun tarvitse olla välttämättä koko ajan saatavilla.

Käyttäjien järjestelmästä saatu kokemus on subjektiivista, mutta palvelutasosopimuksessa on annettu objektiiviset arvot ja lupaukset esimerkiksi saatavuuden tai tehokkuuden suhteen. Kyseisillä määrittelyillä voidaan tutkia objektiivisesti, täyttääkö järjestelmä vaatimukset.

Tang et al. ovat esittäneet, että harvinaisten tilanteiden luomisen avulla pystyttäisiin mittaamaan paremmin korkeaa luotettavuutta ja saatavuutta [11]. Harvinaisten tilanteiden luomisella yritetään varmentua järjestelmän luotettavuudesta esimerkiksi erittäin harvinaisten virhetilojen osalta.

Harvinaisten tilanteiden luominen ja siihen perustuvaan korkean saatavuuden mittaus on kehitettävässä järjestelmässä ongelmallista, sillä järjestelmä ei ole välttämättä tarpeeksi kypsä mitattavaksi. Tang et al:in esityksessä ehdotettiin operationaalisten profiilien, tärkeysotannan, rasitustestauksen ja mittaukseen perustuvan arvioinnin yhdistelmää korkean luotettavuuden ja saatavuuden arvioimiseksi [11].

Tässä työssä pyritään mittaamaan korkeaa saatavuutta useiden eri testausmenetelmien avulla. Työssä keskitytään kehitettävän kriittisen järjestelmän saatavuuden mittaukseen, jolloin mittaus tehdään oikeellisten syötteiden avulla. Niiden lisäksi suoritetaan haavoittuvuustestausta satunnaisuuden perusteella, koska järjestelmän

käyttäjän tekemät päätökset ja toimet eivät ole ennakoitavissa. Perinteisesti haavoittuvuustestauksessa syötetään virheellisiä syötteitä ja tutkitaan toimiiko järjestelmä toivotulla tavalla.

Korkean saatavuuden testaamisessa sovelletaan osittain mallipohjaista testausta sekä toiminnallista testausta ja haavoittuvuustestausta. Toiminnallisessa testauksessa testataan järjestelmälle asetettuja vaatimuksia eli tutkitaan, toimiiko järjestelmä vaatimusten mukaisesti.

Ennen kehitettävän korkean saatavuuden testipedin rakenteen ja testausmenetelmien esittelyä käydään läpi kohdejärjestelmän rakennetta, sekä tämän työn motiivia.

2.2 Automaattinen testaus

Pitkäkestoiset ja kuormaa aiheuttavat testit ovat olennaisessa osassa korkean saatavuuden mittaamisessa, jolloin hyödynnetään automaattista testausta. Pitkäkestoisia ja kuormittavia testejä ei ole mahdollista suorittaa manuaalisella testauksella järkevällä työpanoksella. Automaattisen testauksen yhteydessä voidaan mitata luotettavuutta, tehokkuutta, skaalautuvuutta ja saatavuutta. Automaattisen testauksen hyödyt ja haitat on otettava huomioon ennen testaustyökalun valitsemista tai kehittämistä.

Lewis painottaa, että testaustyökalun valinta tulee tehdä testauksen tavoitteiden perusteella. Testaustyökalun soveltuvuutta täytyy tutkia, kun manuaalinen testaus on riittämätön tavoitteiden täyttämiseen. Lewis käyttää esimerkkinään rasitustestausta: ryhmä testaajia voisi manuaalisesti aiheuttaa kuormaa järjestelmälle, mutta sillä ei saavutettaisi tarkkaa testien toistoa nopeasti ja tehokkaasti eikä manuaalinen testaus olisi kovin kustannustehokas. [12]

2.2.1 Hyödyt

Automaattisen testauksen tärkeimmät hyödyt ovat resurssien säästäminen ajan ja rahan suhteen sekä testaajien kuorman vähentäminen. Automaattisella testauksella voidaan saavuttaa sellaisia tavoitteita, joihin manuaalinen testaus ei kykenisi. Samalla voidaan varmentua siitä, että automaattinen testaus suorittaa ja mittaa testit joka kerta samalla tavalla.

Lewis täydentää automaattisen testauksen hyödyiksi muun muassa nopeuden, suorituksen riippumattomuuden testaajan toimista, testien uudelleenkäytettävyyden, ja luotettavan testien suorituksen vakaassa testiympäristössä. Joillakin testaustyökaluilla on mahdollista mitata koodikattavuutta. [12]

Luotettavuutta vaativien järjestelmien kehityksessä automaattisesta testauksesta on paljon hyötyä: se voi parantaa testauksen laatua eri osa-alueilla, sekä vähentää testauksen järjestelyyn liittyviä toimenpiteitä ja siihen kuluvaa aikaa. Dustin et al.

esittelevät automaattisen testauksen nopeutta verrattuna manuaaliseen testaukseen Quality Assurance Instituten teettämän tutkimuksen pohjalta. Taulukossa 2.2 on kyseisen tutkimuksen tulosvertailu. [13, s. 37, 39-50]

Taulukko 2.2: Quality Assurance Instituten manuaalisen ja automaattisen testauksen vertailu [13, s. 50].

Vaihe	**Manuaalinen (tunnit)**	**Automaattinen (tunnit)**	**Ero prosentteina**
Testaussuunnittelu	32	40	-25 %
Testien määrittely	262	117	55 %
Testien suorittaminen	466	23	95 %
Tulosten analysointi	117	58	50 %
Virheiden dokumentointi	117	23	80 %
Raportointi	96	16	83 %
Kokonaiskesto	1090	277	75 %

Taulukon 2.2 perusteella voidaan todeta, että automaattisesta testauksesta on erittäin paljon hyötyä nopeuden suhteen. Sopivan testaustyökalun valitseminen on haastavaa, ja se voi vaikuttaa testausnopeuteen merkittävästi. Taulukossa ei huomioida tilannetta, jossa testaustyökalu joudutaan kehittämään itse, mikä suurella todennäköisyydellä hidastaa automaattisen testauksen aloitusta ja testausta alkuvaiheessa.

2.2.2 Haasteet

Hyötyjen lisäksi on tarkasteltava automaattisen testauksen tuomia haasteita kattavasti. Näin voidaan ymmärtää, mitkä asiat on huomioitava ennen kuin korkean saatavuuden automaattista testausta voitaisiin harkita ja millä tasolla testausta tulisi suorittaa. Jos testaukseen kyettäisiin varaamaan äärettömän määrän resursseja, ei se välttämättä takaa täyttä hyötyä, vaan testauksesta voi tulla rasite mille tahansa projektille.

Broekman et al. [7, s. 217] esittävät kolme suurinta riskiä automaattiselle testaukselle, jotka ovat muutokset järjestelmässä, sen rakenteessa tai vaatimuksissa. Nämä muutokset voivat aiheuttaa

- ylimääräisten testien tekemisen
- testitapausten muuttumisen
- testitulosten tarkastuspisteiden muuttumisen
- järjestelmän rajapintojen muuttumisen

- järjestelmän toiminnallisuuden muuttumisen
- järjestelmän tuottamien vasteiden tai signaalien muuttumisen
- järjestelmän alustan vaihtumisen tai
- järjestelmän sisäisen toteutuksen muuttumisen.

Ketterissä ohjelmistoprojekteissa otetaan testaajat ja heidän tarpeensa enemmän huomioon. Collins et al:in tutkimus osoittaa, että ryhmittymällä erikseen testaajiin ja kehittäjiin heikentää automaattisen testauksen tuomaa hyötyä. Tutkimuksessa osoitettiin, että parempia tuloksia saatiin testaajien ja kehittäjien välisellä yhteistyöllä. [14]

Tutkimuksessa tutkittiin kolmea eri lähestymistapaa: aluksi vain testaajista koostunut ryhmä suunnitteli testitapaukset, teki osasta testitapauksista automaattisia testejä, suoritti testit, raportoi testeissä havaitut virheet ja suoritti testit uudelleen regressiotestauksen muodossa. Toisessa lähestymistavassa vain kehittäjistä koostunut ryhmä kävi saman prosessin läpi, ja viimeisessä lähestymistavassa testaajat ja kehittäjät tekivät yhteistyötä käyden läpi saman prosessin. [14, s. 442-443]

Tutkimuksessa havaittiin, että kehittäjien ja testaajien yhteistyöllä saatiin eniten hyötyä automaattisen testauksen suunnittelussa, testien suorittamisessa ja samalla parannettiin muiden osallistujien tietämystä testattavasta järjestelmästä. Automaattisen testauksen ylläpitoa voidaan parantaa sillä, että työkalu on kaikkien osapuolien hallittavissa. Kaikkien pitäisi pystyä kirjoittamaan testitapaukset, ajamaan niitä työkalulla ja analysoimaan testeistä saatuja tuloksia. Tutkimuksessa havaittiin, että rasitus- ja tietoturvatestauksen automatisoiminen varhaisessa vaiheessa vähensi lisätyön ja riskien määrää. [14, s. 443-445]

Craig et al. lisäävät automaattisen testauksen uhkina automaattisen testauksen keskittymisen vääriin asioihin ja väärän testaustyökalun valinnan. Automaattisen testauksen hyöty on mitätön käytettyihin resursseihin nähden, jos testauskohdetta testataan vain kerran tai sen toteutus muuttuu jatkuvasti. [15, s. 221]

Craig et al. [15, s. 219-228] mainitsevat useita uhkia automaattisen testauksen osalta:

- Testaukselta puuttuu selkeä strategia.
- Liian suuret odotukset testauksesta.
- Valittuun testaustyökaluun kohdistuvat epäilykset.
- Vajaa koulutus.
- Väärien asioiden testaaminen.

- Väärän työkalun valinta.
- Työkalun helppokäyttöisyyden käyttäminen kriteerinä.
- Väärän testaustyökalun toimittajan valinta.
- Epävakaa työkalu.
- Liian nopea uuden työkalun käyttöönotto.
- Resurssien aliarviointi.
- Puutteellinen tai ainutlaatuinen testiympäristö.
- Testauksen ajoittaminen huonosti.
- Työkaluihin liittyvät kustannukset.

Kaikkia aiemmin luetteloituja uhkia voidaan vähentää huomioimalla ketterän ohjelmistokehityksen tuomat edut testaukseen. Kehitettävän järjestelmän automaattiseen testaukseen liittyvät riskit vähenisivät, kun kehittäjät ja testaajat tekevät yhteistyötä.

Korkean saatavuuden testaamisessa on otettava kaikki aiemmin mainitut uhkat tarkkaan huomioon. Automaattisen testauksen työkalun kehittäminen vain korkean saatavuuden mittaamista varten on erittäin suuri riski mahdollisiin hyötyihin nähden. Uuden työkalun kehittäminen poistaa osan riskeistä, kuten esimerkiksi väärien asioiden testaamisen, väärän työkalun tai toimittajan valitsemisen, sekä monia muita riskejä.

2.2.3 Korkean saatavuuden vaatimukset testaustyökalulle

Korkean saatavuuden mittaamiseksi on hyvä pohtia, mitä vaatimuksia se asettaa testaustyökalulle. Aiemmin todettiin, että saatavuus on olennainen laatutekijä korkeaan saatavuuteen pyrkivissä järjestelmissä luotettavuuden, skaalautuvuuden, ja tehokkuuden lisäksi. Edellä mainittujen laatutekijöiden mittaaminen asettaa vaatimuksen, että testejä on pystyttävä suorittamaan mahdollisimman pitkään ja luotettavasti. Näin voidaan varmistua siitä, että järjestelmä kykenee palvelemaan suurimman osan ajasta — pois lukien ennalta sovitut huoltokatkot.

Kaikkea ei ole mahdollista testata automaattisesti, jolloin testauksessa on otettava huomioon oleellisimmat ja käytetyimmät toiminnot. Testien suoritusjärjestyksen variointi on oltava mahdollista, koska tarkan järjestyksen suunnittelu vie aikaa. Samalla testattavan järjestelmän toimintaa voidaan tutkia luotettavammin, kun testejä

ei suoriteta ennalta määritellyn järjestyksen mukaan. Oleellisimmat ja käytetyimmät toiminnot testattavassa järjestelmässä ovat tärkeässä osassa korkean saatavuuden testauksessa.

Onnistuneiden testien lisäksi testattavan järjestelmän tehokkuutta tulisi tarkastella — hidas ja luotettava järjestelmä ei välttämättä ole käyttäjän mielestä tarpeeksi saatavissa. Tehokkuuteen on pakko kiinnittää huomiota järjestelmän eri osien osalta. Järjestelmän tilaa tulee seurata ja samalla kerätä tietoa järjestelmästä testauksen aikana.

Raportointiominaisuus ei ole ehdoton vaatimus testien suoritusta ajatellen, vaan sen tarkoitus on palvella testaajien lisäksi myös muita osapuolia. Raportoinnilla voidaan tarkastella testien tuloksia, ja sillä voidaan tuoda lisätietoa suoritetuista testeistä. Testien tuloksista täytyy näkyä selkeästi suoritettiinko testit onnistuneesti ja kuinka kauan testien suoritukseen meni aikaa. Epäonnistuneiden testien osalta on saatava lisätietoa, miksi testit epäonnistuivat, jotta järjestelmässä olevia virheitä pystytään korjaamaan.

Tietoturva on oleellinen osa kriittisiä järjestelmiä, jota ei voi jättää huomiotta. Mahdolliset tietoturva-aukot voivat vaikuttaa luotettavuuden ja saatavuuden heikkenemiseen. Haavoittuvuustestausta tulisi hyödyntää korkean saatavuuden testauksessa: injektioiden ja muiden haavoittuvuustestauksen tekniikoita voidaan hyödyntää muun testauksen ohella.

Korkean saatavuuden mittaaminen vaatii kestotestauksen lisäksi kuormituksen tuottamista. Klusteroinnin hyödyntäminen kuorman tuottamiseen on hyvä tapa selvittää järjestelmän kestävyys eri kuormitustilanteissa. Testauksen klusteroinnin avulla voidaan tuottaa realistista kuormaa samalle järjestelmälle.

2.3 Suorituskykytestaus

Järjestelmän tehokkuutta voidaan mitata käyttäjänäkökulmasta: miten nopeasti järjestelmä vastaa käyttäjän aiheuttamaan vasteeseen? Tehokkuuden mittaamiseen ei ole yhteistä standardia, vaan termi tehokkuus määritellään palvelutasosopimuksissa ja samalla luvataan asiakkaalle tietty suorituskykytaso [3, s. 3]. Tehokkuus on saatavuuden ohella subjektiivinen mittari: jonkun henkilön mielestä huono suorituskyky voi olla riittävä toiselle henkilölle.

Suorituskykytestauksessa (engl. performance testing) selvitetään järjestelmän kykyä selviytyä tehtävistään vaaditussa ajassa erilaisissa tilanteissa [3, s. 2]. Suorituskykytestaus voidaan jakaa erilaisiin testityyppeihin, jotka mittaavat tehokkuutta eri perspektiiveistä.

Broekman et al:in mukaan käyttötiloihin perustuvaa testausta (engl. statistical usage testing) voidaan hyödyntää suorituskykytestauksessa. Kyseinen testaustapa huomioi järjestelmään liittyviä asioita ja niistä voidaan muodostaa mahdollisimman

realistisia käyttötapauksia. Tehokkuutta voidaan testata ja mitata kaikissa sellaisissa tilanteissa, joita todennäköisemmin ilmenee järjestelmän normaalin käytön yhteydessä. [7, s. 51, 158]

Työkalulle esitetyissä vaatimuksissa todettiin, korkean saatavuuden testaamiseksi on otettava huomioon kuormittavia testejä. Kuormittavia testausmenetelmiä ovat kesto-, rasitus-, kuormitus-, ja volyymitestaus. Tässä työssä pyritään hyödyntämään vain kestotestausta, koska rasitus ja kuormitustestaus vaativat testattavalta järjestelmältä stabiiliutta kehityksen ja toiminnallisuuden suhteen, jotta testauksesta saataisiin parasta hyötyä. Volyymitestaus jätetään kokonaan työn ulkopuolelle, koska se ei sovellu korkean saatavuuden mittaukseen käyttäjänäkökulmasta ja rasitustestauksessa pyritään mittaamaan osittain myös volyymia.

2.3.1 Kestotestaus

Kestotestauksessa (engl. endurance/soak testing) testejä suoritetaan jatkuvasti pitkällä aikavälillä ja järjestelmä kuormittuu vakiokuormalla. Kestotestauksella pyritään löytämään testattavasta järjestelmästä esimerkiksi muistivuotoja tai mahdollisia raja-arvoja ajan ja luotettavuuden suhteen. [3, s. 39]

Kestotestaus on olennaisin osa korkean saatavuuden mittauksessa, koska sillä pyritään osoittamaan, että järjestelmä kykenee suoriutumaan tehtävistään mahdollisimman pitkään vakioidulla kuormalla. Korkeaa saatavuutta voidaan mitata tarkasti ja luotettavasti vain pitkällä aikavälillä. Kestotestauksella pyritään mittaamaan, kuinka kauan järjestelmä kykenee toimimaan luotettavasti.

2.3.2 Kuormitustestaus

Kuormitustestauksessa (engl. load testing) suoritetaan testejä tietyllä kuormalla ennalta määritellyn ajan verran. Kuormitustestauksella on selkeä keskeytyspiste: testaus lopetetaan, jos tietyn määrä testejä epäonnistuu tai aikaraja ylittyy. Molyneaux määrittelee termin seuraavasti: kuormitustestaus on testausmenetelmä, jolla pyritään kuormittamaan järjestelmää realistisella kuormalla [3, s. 39].

Kuormitustestauksella pyritään löytämään sellaisia tehottomuutta aiheuttavia ja toiminnallisia virheitä, joita ei löydettäisi manuaalisella testauksella tai kestotestauksella. Kuormitus- ja rasitustestauksen aikana on mahdollista havaita esimerkiksi säielukkoihin liittyviä ongelmia. Tehokkuuteen liittyvät ongelmat ilmenevät testattavan järjestelmän toimintojen vasteaikojen heikentymisenä tai testien suoritusten hidastumisena.

Kuormitustestausta ei käytetä korkean saatavuuden mittaamiseen tässä työssä, koska siitä ei saada tarpeeksi hyötyä testauskohteena olevan kehitettävän järjestelmän takia. Kuormitustestausta on parempi hyödyntää siinä vaiheessa, kun testaus-

kohteen kehitys on stabiloitunut tarpeeksi. Myöhemmin on tavoitteena hyödyntää kuormitustestausta korkean saatavuuden mittaamiseen ja siitä esitetään jatkokehitysajatuksia kohdassa 4.5.3.

2.3.3 Rasitustestaus

Rasitustestauksessa (engl. stress testing) pyritään etsimään järjestelmän ylärajaarvoja kuormittamalla järjestelmää mahdollisimman suurella kuormalla. Ylärajaarvoksi saadaan sellainen kuorma, jolla järjestelmä pääsee juuri ja juuri testeistä läpi. Molyneaux lisää, että rasitustestausta suoritetaan niin kauan, kunnes jokin testattavassa järjestelmässä rikkoutuu. Esimerkiksi sisäänkirjautuminen epäonnistuu tai vasteajat nousevat hyväksymättömälle tasolle. [3, s. 39]

Melzer luokittelee rasitustestauksen kahteen alatyyppiin: pitkä- ja lyhytkestoiseen. Jälkimmäinen on perinteinen rasitustestauksen muoto — järjestelmää pyritään kuormittamaan erittäin suurella kuormalla lyhyen ajan sisään. Pitkäkestoisessa rasitustestauksessa testaus suoritetaan suurella vakiokuormalla pitkällä aikavälillä. [16]

Kuormitustestauksen tavoin, tätä testausmenetelmää ei käytetä tässä työssä korkean saatavuuden mittaamiseen, koska on parempi keskittyä kestotestaukseen. Rasitustestausta on tarkoitus hyödyntää myöhemmin ja siitä on jatkokehitysajatuksia kohdassa 4.5.4.

2.4 Mallipohjainen testaus

Mallipohjaisessa testauksessa (engl. model-based testing) tehdään esimerkiksi testattavaa järjestelmää kuvaavia malleja, joiden perusteella testaustyökalu luo suoritettavia testejä. Utting et al. mainitsevat myös muita mallipohjaiseen testauksen variaatioita, kuten testidatan luonnin. [6, s. 6-7]

Tässä työssä kokeillaan mallipohjaisen testauksen soveltuvuutta kestotestauksen kanssa. Mallipohjaisessa testauksessa pyritään hyödyntämään testattavan järjestelmän tiloja mallinnuksessa. Testitapauksiin määritellään tietyt odotusarvot, joista testaustyökalu voi päätellä, onko testi onnistunut. Yksinkertaisimmillaan testimalliin kuvataan kaikki testattavan järjestelmän tilat ja tiloihin liittyvät testitapaukset. Esimerkiksi testattavan järjestelmän käyttöliittymästä voidaan määritellä eri tilat ja testitapaukset voidaan määritellä käyttäjän mahdollisista toiminnoista.

Mallipohjainen testaus jakaantuu kahteen eri kategoriaan: online- ja off-line-testaukseen. Off-line-testauksessa luodaan suoritettavat testit mallien mukaan mallipohjaisen testaustyökalun avulla ennen testien suoritusta. Online-testauksessa testit luodaan suorituksen aikana: testaustyökalu lukee mallia, luo ja suorittaa yhden testin kerrallaan sekä tuottaa testin tuloksen. Tuloksen jälkeen testaustyökalu valit-

see uuden testin suoritettavaksi. Online-mallipohjainen testaus kytkeytyy suoraan testattavaan järjestelmään ja pyrkii testaamaan järjestelmää dynaamisesti. [6, s. 28]

Mallipohjaisia testaustyökaluja on olemassa useampia; ne ovat yleensä kehittyneet tietyn sovellusalueen vaatimuksien pohjalta. Kuitenkin Puolitaival et al. lisäävät, että oman testaustyökalun kehittäminen on hyödyllisempää, jos mikään olemassa olevista työkaluista ei täytä testauksen tarpeita. [17]

Tilastollinen mallipohjainen testaus (engl. model based statistical testing) on tyypillisesti online-mallipohjaisen testauksen erikoismuoto, jossa hyödynnetään valittua algoritmia sekä operationaalisia malleja testien generointiin ja suorittamiseen. Operationaalinen profiili on kvantitatiivinen jäsennelmä, miten järjestelmää käytetään. Operationaaliset profiilit perustuvat järjestelmän tapahtumien todennäköisyyteen, eli ne kuvaavat kaikki mahdolliset tilat ja tapahtumat sekä tilojen välisiä yhteyksiä.

Tässä työssä käytetään tilastollista mallipohjaista testausta, hyödyntäen operationaalisia profiileja. Näin voidaan mallintaa tarkasti erilaisia käyttötapoja etukäteen, ja testaustyökalu generoi malleista suoritettavia testejä ja suorittaa ne dynaamisesti. Testejä voidaan varioida testaajan mielen mukaan ja testitapauksiin voidaan liittää erilaisia käyttäjään perustuvia toimintamalleja sekä syötteitä, unohtamatta järjestelmää kuvaavia malleja.

Työssä hyödynnetään mallipohjaista testausta harmaalaatikko-, haavoittuvuustestauksen ja toiminnallisen testauksen yhteydessä. Korkean saatavuuden mittaamiseksi on tehtävä hyväksyttäviä ja virheellisiä testisyötteitä, joiden avulla voidaan tutkia miten järjestelmä käyttäytyy eri tilanteissa.

2.5 Operationaaliset profiilit

Broekman et al. mukaan Musa [18] esitteli konseptin operationaalisista profiileista. Näiden avulla voidaan koota erilaisia käyttäjään perustuvia toimintamalleja, joiden perusteella testaustyökalu valitsee tilanteeseen sopivan testitapauksen suoritettavaksi. [7, s. 158-159]

Operationaalisen profiilin koostaminen vaatii testattavaan järjestelmään liittyvien tekijöiden tarkan analysoinnin. Broekman et al. [7, s. 159] jakaakin analysoinnin viiteen osaan:

1. Asiakkaiden analysointi.
2. Käyttäjien analysointi.
3. Järjestelmän tilojen analysointi.
4. Toiminnallisten profiilien (engl. functional profiles) analysointi.
5. Operationaalisten profiilien analysointi.

Asiakkaiden analysoinnissa otetaan huomioon eri asiakkaat sekä heidän toimintatavat ja tarpeet. Jokaisesta asiakastyypistä luodaan asiakasprofiili ja kaikista asiakasprofiileista valitaan tärkeimmät profiilit, joita hyödynnetään käyttäjien analysoinnissa. [7, s. 159]

Käyttäjien analysoinnissa koostetaan käyttäjäprofiileja asiakasprofiilien pohjalta. Käyttäjäprofiili kuvaa kyseisen käyttäjätyypin toimintaa tarkasti. Kuvauksen lisäksi voidaan määritellä, kuinka tärkeä tai yleinen kyseinen toimintatapa on. [7, s. 160]

Järjestelmän tilaprofiili koostuu joukosta samantyyppisistä toiminnoista. Toiminnot perustuvat käyttäjäprofiileihin, ja toiminnot tulee määritellä siten, että ne eroavat merkittävästi toisistaan. [7, s. 160]

Toiminnalliset profiilit muodostetaan järjestelmän tilaprofiileissa määritellyistä toiminnoista. Toimintojoukot analysoidaan ja jokaiselle joukolle arvioidaan toteutumistodennäköisyys. [7, s. 160]

Viimeiseksi määritellään operationaaliset profiilit. Jokaiselle toiminnalliselle profiilille luodaan operationaalinen profiili, joka kuvaa toiminnallisen profiilin käyttötarkoitusta. Broekman et al. [7, s. 160] määrittelevät, että operationaalisen profiilin kuvaus vastaa kysymyksiin:

- Milloin järjestelmä on *tässä* tilassa?
- Millä todennäköisyydellä *tämä* on seuraava tapahtuma, joka tapahtuu käyttäjän toimesta?

Operationaaliset profiilit määrittelevät kaikki yhdistelmät järjestelmän tiloista ja tiloihin liittyvistä tapahtumista. Taulukossa 2.3 on esimerkki operationaalisesta profiilista.

Taulukko 2.3: Esimerkki operationaalisesta profiilista.

	Tapahtuma 1	**Tapahtuma 2**	...	**Tapahtuma m**
Tila 1	$P_{1,1}$	$P_{1,2}$	...	$P_{1,m}$
Tila 2	$P_{2,1}$	$P_{2,2}$	...	$P_{2,m}$
...	...	...	...	...
Tila n	$P_{n,1}$	$P_{n,2}$	...	$P_{n,m}$

Taulukossa 2.3 käytetään lyhennettä P, joka vastaa seuraavan mahdollisen tapahtuman todennäköisyyttä tilan suhteen. Tapahtuma edustaa käyttäjän aiheuttamaa tapahtumaa, ja tila kuvaa järjestelmän tilaa.

2.6 Harmaalaatikkotestaus

Testitapausten kehittämisessä hyödynnetään harmaalaatikkotestausta (engl. gray-box testing), joka on mustalaatikko- ja lasilaatikkotestauksen yhdistelmä [2, s. 207-209]. Harmaalaatikkotestaus sisältää molempien testausmenetelmien hyvät puolet:

testattavan järjestelmän ulkoiset ja sisäiset tekijät voidaan ottaa huomioon testien suunnittelussa.

Mustalaatikkotestauksessa (engl. black-box testing) järjestelmää testataan syötteiden ja vasteiden avulla, kun lähdekoodia ei ole saatavilla. Mustalaatikkotestaus perustuu järjestelmän testaamiseen ulkopuolisen näkökulmasta, jolloin testaajan ei tarvitse välittää järjestelmän sisäisestä toiminnasta lainkaan. Mustalaatikkotestaus soveltuu eri testausvaiheisiin ja testejä voidaan suorittaa helposti uudelleen. Mustalaatikkotestauksen huonoja puolia ovat, että testituloksia voidaan arvioida väärin ja kaikkia järjestelmän ominaisuuksia ei voida testata. Mustalaatikkotestauksessa ei tutkita järjestelmän sisäistä toimintaa, jolloin virheille ei välttämättä löydy selkeää syytä ja virheen sijaintia on vaikea jäljittää. [2, s. 56-57]

Lasilaatikkotestauksessa (engl. white-box testing) testaus perustuu saatavilla olevaan lähdekoodiin — testauksessa voidaan keskittyä suoraan koodissa havaittuihin riskialttiisiin paikkoihin. Lasilaatikkotestauksen hyvät puolet ovat sen soveltuvuus ohjelmistokehityksen rinnalla ja ajankulutus. Huonona puolena on koodin välttämättömyys testauksessa. Jos testattavassa järjestelmässä on paljon koodia, ei lasilaatikkotestausta voi hyödyntää tehokkaasti integraatio- ja järjestelmätestaustasolla. Patton huomauttaa, että testaaja voi liian helposti tehdä testitapauksia suoraan koodin toiminnan perusteella. Silloin kyseisen toiminnon toimivuutta ei välttämättä testata tarpeeksi kattavasti. [2, s. 56, 94-95]

Harmaalaatikkotestausta hyödynnetään korkean saatavuuden testauksen määrittelyssä, koska tässä työssä keskitytään järjestelmätestaukseen. Yksittäiset testitapaukset voidaan tarvittaessa kehittää suoraan järjestelmän lähdekoodiin perustuen, unohtamatta graafisen käyttöliittymäkirjaston tuomaa rajapintaa.

2.7 Toiminnallinen testaus

Toiminnallinen testaus on tyypillinen mustalaatikkotestauksen muoto, jonka tarkoituksena on testata järjestelmälle asetettuja vaatimuksia [2]. Tällöin pyritään selvittämään, toimiiko järjestelmä asetettujen toiminnallisten vaatimusten mukaisesti. Toiminnallisessa testauksessa käytetään vain hyväksyttäviä syötteitä eikä keskitytä aiheuttamaan virhettä testattavaan järjestelmään virheellisten syötteiden avulla.

Toiminnallista testausta voidaan hyödyntää korkean saatavuuden mittauksen osalta monipuolisesti. Esimerkiksi voidaan määritellä tiedon muokkaamiseen ja lukemiseen liittyviä testejä, ja niiden avulla aiheuttaa kuormaa testattavaan järjestelmään useiden testien suorittajien voimin. Oikeelliset syötteet eivät takaa sitä, että järjestelmä toimisi järkevästi suuren kuorman vaikutuksesta.

Tässä työssä ei ole kuitenkaan tarkoitus käydä kaikkia testattavan järjestelmän toimintoja läpi ja testata niitä, vaan keskitytään vain keskeisiin toimintoihin. Kuormaa aiheuttavien toimintojen testaamisella voidaan pyrkiä mittaamaan korkeaa saa-

tavuutta. Uusia testitapauksia voidaan suunnitella, kun keskeisiä toimintoja on testattu tarpeeksi.

2.8 Haavoittuvuustestaus

Mustalaatikkotestaukseen perustuvan haavoittuvuustestauksen (engl. fuzz testing) idea on päinvastainen kuin toiminnallisella testauksella: järjestelmään syötetään virheellisiä syötteitä ja järjestelmään pyritään aiheuttamaan häiriötä kaikin keinoin [1, s. 22]. Testattavaa järjestelmää ei voi luokitella saavutettavaksi, jos sen toiminta häiriintyy haavoittuvuustestauksen seurauksena. Tietynlainen testijärjestys voi myös aiheuttaa järjestelmään häiriötä, joten senkin mahdollisuus on hyvä ottaa huomioon testattavassa järjestelmässä.

Testiympäristön vaikutukset järjestelmään on otettava huomioon. Järjestelmä voi toimia yhdessä ympäristössä vakaasti ja täysin halutulla tavalla, mutta se ei takaa sitä, että järjestelmä toimisi täysin samalla tavalla toisessa ympäristössä.

Virheellisten syötteiden määrä on poikkeuksetta suurempi kuin hyväksyttävien syötteiden, mikä on haavoittuvuustestauksen huono puoli. Kriittisten järjestelmien kehityksessä tulee ottaa huomioon mahdollisten virheiden vaikutus järjestelmän toimintaan ja vakauteen. Järjestelmän haavoittuvuutta pyritään yleensä estämään käyttöliittymätasolla, jolloin tarkistetaan erilaiset syötteet, jotka voisivat olla potentiaalisia uhkia ja ne eristetään tarpeen mukaan.

Haavoittuvuustestauksen perinteinen testaustapa on raakaan voimaan (engl. brute force) perustuva väsyttämistekniikka, jossa järjestelmää kuormitetaan jatkuvasti virheellisillä syötteillä. Schieferdeckerin mukaan haavoittuvuustestausta voitaisiin hyödyntää oikeellisten syötteiden kanssa mallipohjaisen testauksen avulla. Schieferdeckerin mielestä haavoittavien syötteiden sekoittaminen oikeellisten kanssa voi johtaa uusien virheiden löytämiseen ja parantaa järjestelmän turvallisuutta. [19]

Injektiot ja muut tietoturva-aukot ovat mahdollisia kaikissa järjestelmissä, joissa tietoturvallisuuteen ei ole panostettu. Erilaiset ohjelmistovirheet voivat antaa hyökkääjälle mahdollisuuden hyödyntää virheiden aiheuttamia tietoturva-aukkoja hyökkäykseen. Tyypillisimpiä uhkia ovat tietokantaan tai järjestelmään kohdistetut injektiot eli käyttäjä tai hyökkääjä voi syöttää sellaisia syötteitä, jotka sisältävät komentoja. Injektioiden avulla pyritään järjestelmän toimintaa häiritsemään tai saamaan hyökkääjälle tärkeitä tietoja järjestelmästä. Vuonna 2011 CWE:n tehdyn raportin mukaan, vaarallisimpia ohjelmistovirheitä ovat tietokantainjektiot, käyttöjärjestelmään kohdistuvat injektiot sekä järjestelmän muistivuodot [20].

Haavoittuvuustestauksen suorittamisen edellytyksenä on testattavan järjestelmän kypsyys — haavoittuvuustestausta ei ole mielekästä suorittaa, jos järjestelmä ei toteuta kaikkia suunniteltuja toiminnallisuuksia tai järjestelmän normaali toiminta aiheuttaa ongelmia. Haavoittuvuustestausta voidaan suorittaa siinä vaiheessa, kun

järjestelmän kehitys ja virheiden määrä on vakaantunut selvästi. Tietoturvatestauksen tarkoituksena on keskittyä haavoittuvuuksien jäljittämiseen, jolloin haavoittuvuustestauksesta ei ole välttämättä hyötyä kuormitustesteissä.

2.9 Testattavien järjestelmien kuvaus

Ennen kehitettävän korkean saatavuuden testipedin rakenteen esittelyä kuvataan testauskohteet, jotta saadaan selville tämän työn motiivi. Testauskohteina ovat uusi kehitettävä hätäkeskustietojärjestelmä ja sen osana toimiva Varotietopalvelu.

Hätäkeskustietojärjestelmä ERICA korvaa nykyisen hätäkeskustietojärjestelmän vuonna 2015, ja siltä vaaditaan toimintavarmuutta sekä vähintään 99,996 % saatavuutta [21]. Näiden vaatimuksien takia korkean saatavuuden testipedillä pyritään tutkimaan, onko kehitettävä järjestelmä luvattujen vaatimuksien mukainen. Mittauksessa etusijalle nousevat aiemmin mainitut laatutekijät eli luotettavuus, saatavuus, tehokkuus, skaalautuvuus ja ylläpidettävyys.

Uusi järjestelmä tulee koostumaan hätäkeskustietojärjestelmäpalvelusta, hätäkeskuspäivystäjän sovelluksesta, sekä muista näihin liittyvistä liitynnöistä, laitteista, sovelluksista ja palveluista. Palveluihin lukeutuu muun muassa Varotietopalvelu, joka on varotietojen hallintaan tarkoitettu selainpohjainen sovellus, jolla voidaan luoda, muokata, tarkastella, hyväksyä ja poistaa varotietoja. Varotieto voi olla henkilö, osoite tai kulkuneuvo. Tehtävän välittämisen yhteydessä lisä- ja tukitiedoissa annetaan tehtävän suorittajalle työturvallisuuden edellyttämät tarpeelliset käytettävissä olevat tiedot, sekä tehtävää muuten tarkentavat lisätiedot. Esimerkiksi varotiedot näytetään ilmoituslomakkeella automaattisesti, kun hätäkeskuspäivystäjä täyttää ilmoitusta ja tietoihin sopiva varotieto löytyy.

Hätäkeskustietojärjestelmän pääasiallinen käyttö tapahtuu hätäkeskuspäivystäjän sovelluksesta, joka sisältää käyttöliittymän, johon käyttäjän tarvitsemat toiminnot on integroitu roolipohjaisesti. Hätäkeskuspäivystäjä voi muun muassa kirjautua järjestelmään roolipohjaisesti, vastaanottaa ilmoituksia, sekä kommunikoida eri kenttäyksiköiden kanssa.

Hätäkeskustietojärjestelmäpalvelu toimii koko hätäkeskustietojärjestelmän ytimenä käsittäen muun muassa hätäilmoitusten vastaanoton, tehtävien välittämisen ja seurannan sekä muita palveluja. Jokainen hätäkeskustietojärjestelmän tarjoama palvelu tarjoaa palvelunsa muiden palveluiden ja käyttöliittymien käytettäväksi.

Hätäkeskustietojärjestelmäpalvelu tarjoaa esimerkiksi yleisen hätäilmoitusjonopalvelun, jossa kaikki järjestelmään saapuvat hätäilmoitukset tuodaan yhteen keskitettyyn hätäilmoitusjonoon. Uudella järjestelmällä pyritään siihen, että esimerkiksi ruuhkatilanteessa voidaan ohjata hätäilmoituksia toiseen keskukseen ilman merkittävää viivettä.

Tässä työssä pääpaino on testipedin kehityksessä sekä siinä, että voidaanko korke-

aa saatavuutta mitata sillä. Hätäkeskustietojärjestelmä on hyvä esimerkki kriittisestä järjestelmästä ja täten oivallinen testikohde tässä tutkimuksessa sekä testipedin kehityksessä.

3. KÄYTETYT TESTAUSTEKNIIKAT, -MENETELMÄT, JA -AINEISTOT

Automaattisen testauksen oleellisin tehtävä on suorittaa testejä ja mahdollisesti tuottaa ymmärrettäviä testituloksia sekä tarpeen vaatiessa suorittaa testit uudelleen. Automaattinen testaus toimii manuaalisen testauksen täydentäjänä, jolloin testaaja voi keskittyä oleellisiin työtehtäviin ja raportoida tulokset eteenpäin.

Craig et al:in mukaan testaustyökalut eivät kykene vastaamaan kaikkiin haasteisiin eivätkä välttämättä tuo oikeita vastauksia testaajien kohtaamiin ongelmiin. Oikean testaustyökalun valinta tuo lisäarvoa ja tehokkuutta testaajien työhön. [15]

3.1 Testaustyökalun valinta

Kohdassa 2.2.3 esitettiin vaatimuksia korkean saatavuuden testaukselle. Korkean saatavuuden mittaus ja sen testaus asettavat selkeät vaatimukset automaattisen testauksen osalta:

- Pitkäkestoinen testaus — jatketaan testausta niin kauan kuin se on mielekästä suorittaa.
- Testijärjestykseen vaikuttaminen esimerkiksi mallipohjaisen testauksen keinoin.
- Testitulosten raportointi — testien onnistuminen selkeiden ja yksiselitteisten kriteerien nojalla, ja kuinka kauan testien suorituksiin meni aikaa.
- Syy testin epäonnistumiseen.
- Järjestelmän eri osien monitorointi.
- Klusterointiominaisuus — useiden testien suoritus samaan aikaan ja kohdentaminen tiettyyn järjestelmään.

Korkean saatavuuden testausta varten ei löytynyt sopivaa työkalua, joka olisi täyttänyt edelliset vaatimukset, joten testausta varten kehitettiin testipeti. Tämän avulla pyritään testaamaan ja mittaamaan kaikkia korkeaan saatavuuteen liittyviä laatutekijöitä eli saatavuutta, luotettavuutta, skaalautuvuutta, tehokkuutta ja ylläpidettävyyttä.

Seuraavaksi käsitellään testipedin rakennetta, sen tuottamaa tulosaineistoa ja lopuksi määritellään käytetyt testausmenetelmät sekä määrittelijät. Käytetyt testausmenetelmät määritellään korkeaan saatavuuteen liittyvien laatutekijöiden mukaisesti.

3.2 Korkean saatavuuden testipeti

Testipeti kehitettiin testaamaan korkeaa saatavuutta, joka mahdollistaa saatavuuden lisäksi luotettavuuden, skaalautuvuuden, tehokkuuden sekä ylläpidettävyyden mittauksen. Testipedillä voidaan hyödyntää perinteisen automaattisen testauksen lisäksi online-tyyppistä mallipohjaista testausta, jolloin testien generointi tapahtuu dynaamisesti testipedin suorituksen aikana.

Varhaisversio testipedistä hyödynsi yksikkötestaukseen keskittyvän JUnitin kirjastoja, mutta jo seuraavassa testipedin versiossa otettiin ne pois käytöstä, koska ne eivät soveltuneet korkean saatavuuden testaukseen. JUnitin kirjastojen korvaajaksi kehitettiin testilogiikka, joka mahdollistaa dynaamisen testien suorituksen.

Testien määrittelemiseksi testipedissä käytetään puurakennetta eli testijoukot voivat sisältää muita testijoukkoja ja -tapauksia. Kaikkiin testijoukkoihin voidaan määritellä testitapausten lisäksi nimi, kuvaus, kierrosmäärä, ja satunnaisuus. Nimi ja kuvaus ovat testaajan määriteltävissä. Kierrosmäärät ja satunnaisuus vaikuttavat suoritusjärjestykseen. Testipeti luo uuden kokonaisen testikierroksen testijoukkoa varten, kun testijoukkoa on suoritettu tietyn kierrosmäärän verran. Testijoukon satunnaisuudella vaikutetaan testijoukon testitapausten ja -joukkojen suoritusjärjestykseen.

Mallipohjaista testausta hyödynnetään operationaalisten profiilien kanssa, jolloin voidaan mallintaa järjestelmästä erilaisia käyttötapauksia ja testipeti luo niistä dynaamisesti suoritettavia testejä. Testitapauksille ja -joukoille voidaan antaa todennäköisyyksiä, joiden perusteella testipeti valitsee sopivan testin edellisen testin suorittamisen jälkeen. Todennäköisyyksien antamisen lisäksi on mahdollista tehdä testauskohteen muuttujiin perustuvia ehtoja, jolloin esimerkiksi hätäilmoitusjonon koko voi vaikuttaa testin valintaan.

Testitapaus muistuttaa yksikkötestejä eli sillä pyritään testaamaan järjestelmän yksittäistä toimintoa. Korkean saatavuuden testipetissä toiminnon testaamisen lisäksi testitapaus mittaa toiminnon kokonaiskeston eli toiminnon vasteajan. Vasteaika pyritään mittaamaan käyttäjänäkökulmasta alusta loppuun eli toiminnon aloituksesta aina järjestelmän antamaan vasteeseen asti. Vasteajasta voidaan tutkia toiminnon tehokkuutta ja verrata muihin testituloksiin.

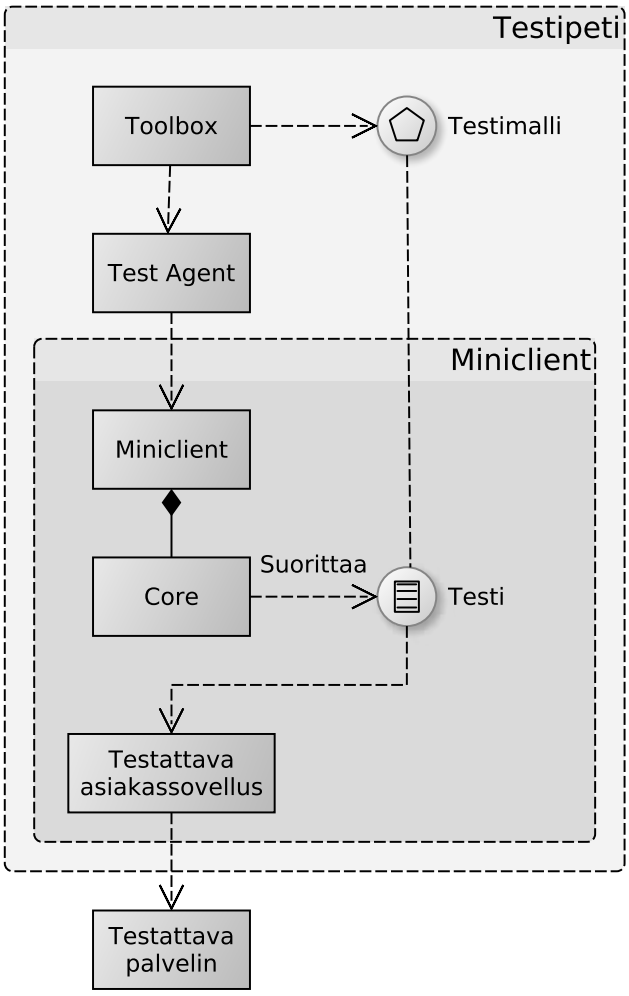
Normaalien testijoukkojen lisäksi voidaan luoda mallipohjaiseen testaukseen erikoistuneita testijoukkoja, jotka määrittelevät todennäköisyydet tai ehdot suoritettaville testeille. Tällöin voidaan vaikuttaa paremmin siihen, millaisia testejä ajetaan,

mutta kuitenkin malli itsessään määrää, mitä testataan. Malliin pohjautuville testijoukoille voidaan asettaa todennäköisyydet tai ehtoja, joissa tarkastellaan tiettyjä muuttujia ja niiden arvoja.

Kuvassa 3.1 on esitetty testipedin rakenne testien suorittamisen näkökulmasta. Kehitetty testipeti koostuu neljästä komponentista: High Availability Toolboxista, High Availability Test Agentista, High Availability Miniclientista ja High Availability Coresta. Kyseisten komponenttien nimet lyhennetään ottamalla High Availability -etuliitteet pois myöhemmissä viittauksissa.

Toolboxilla voidaan tehdä erilaisia testimalleja, joista Core luo dynaamisesti suoritettavia testejä ja suorittaa niitä yksitellen, kuten kuvasta 3.1 näkee. Jokaisella Miniclientilla on oma Core testien suoritusta varten — kuvassa on esitetty erikseen Miniclient-komponetti, joka on toteutettu Miniclient-nimiseen Java pakettiin (engl. Java package). Testattavan asiakassovelluksen toteutus on kiedottu Miniclient-pakettiin, jolloin testejä voidaan kohdistaa suoraan testattavaan sovellukseen.

Toolbox on graafisen käyttöliittymän omaava työkalu testien tekoa ja mallinnus-



Kuva 3.1: Testipedin rakenteen moduulikuva.

ta, tulosten ja järjestelmän tietojen tarkastelua varten. Sillä voidaan nähdä, mitkä Test Agentit ovat käynnissä testaukseen tarkoitetuilla asiakaskoneilla, ja mitkä Miniclientit ovat suorittamassa testejä. Toolboxilla voidaan käynnistää tai sammuttaa valittuja Miniclientteja. Kohdassa 3.2.2 käsitellään Toolboxin ominaisuuksia tarkemmin.

Test Agent on asiakaskoneella toimiva rajapinta Miniclienttien käynnistykseen. Miniclient on testejä suorittava sovellus, joka kietoo testattavan asiakassovelluksen toteutuksen sisäänsä. Core on testipedin ydin, johon on toteutettu testilogiikka. Core luo mallista suoritettavia testejä dynaamisesti ja suorittaa ne, kuten kuvassa 3.1 näkyy. Seuraavassa kohdassa käsitellään tarkemmalla tasolla Miniclientin toimintaa.

Testipedillä voidaan seurata, kuinka kehitettävän järjestelmän laatu muuttuu. Esimerkiksi vasteaikojen mittauksella voidaan tutkia, paraneeko järjestelmän yksittäisen toiminnon suoritus, kun sitä optimoidaan. Jos järjestelmässä suoritettavan toiminnon tehokkuus heikkenee, sitä voidaan pyrkiä parantamaan seuraavaan testiversiota varten. Testipeti tallentaa automaattisesti tietokantaan jokaisen testatun testin tuloksen sekä tietoa testattavan järjestelmän monitoroitavista osista.

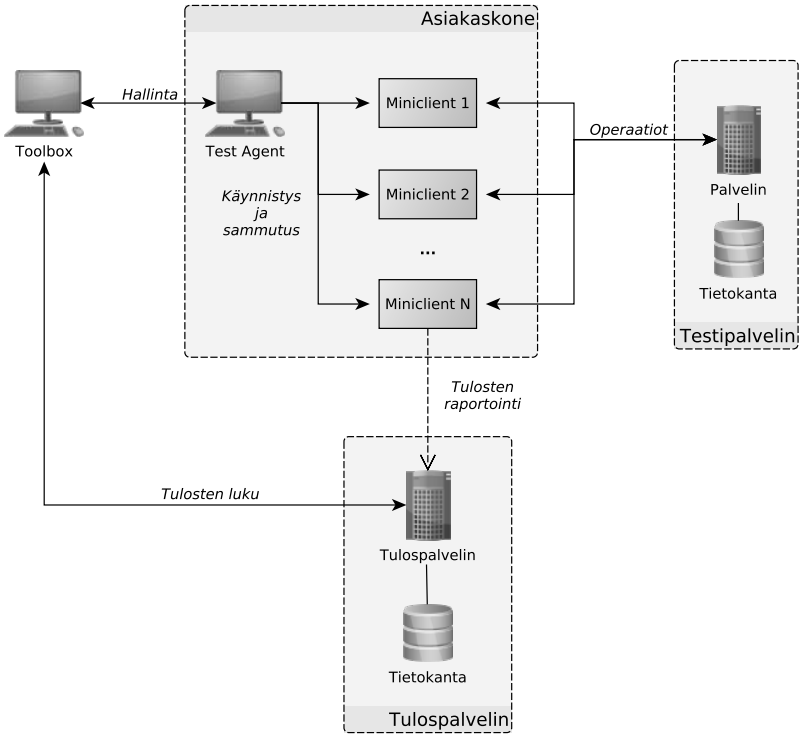
3.2.1 Miniclient

Kehitettäviin järjestelmiin kuuluu yleensä monia eri osia, kuten esimerkiksi asiakas- ja palvelinsovelluksia sekä kolmannen osapuolen tarjoamia tekniikoita, jotka voivat vaikuttaa merkittävästi korkean saatavuuden testaukseen. Miniclientin avulla voidaan keskittyä koko järjestelmän tai valittujen komponenttien korkean saatavuuden testaamiseen.

Miniclienttiin voidaan kietoa koko asiakassovelluksen toteutus tai vaihtoehtoisesti simuloida asiakassovelluksen toimintaa. Miniclient voi silloin käyttää asiakassovelluksen tarjoamia rajapintoja ja sen tietomallia, jotta saadaan riisuttu malli asiakassovelluksesta. Tällä simulointisovelluksella voidaan esimerkiksi kirjoittaa tietoa järjestelmään tai lukea järjestelmästä saatavaa tietoa samalla tavalla kuin oikeasta asiakassovelluksesta. Testitapausten toteuttaja voi hyödyntää asiakassovelluksessa käytettävää graafisen käyttöliittymäkirjaston rajapintaa, jolloin voidaan mitata käyttöliittymästä eri asioita.

Kuvassa 3.2 on esitetty korkeamman tason moduulikuva testipedin eri osien välisistä kommunikaatioista. Moduulikuva esittää esimerkkikokoonpanoa, jossa testataan suoraan palvelinta käyttäen asiakassovelluksen rajapintaa hyödyksi. Testejä suorittavat Miniclientit mittaavat kaikkien testien vasteajat, joilla pyritään mittaamaan käyttäjänäkökulmasta tehokkuutta. Miniclient voi kietoa koko asiakassovelluksen toteutuksen, jolloin muodostuisi abstrakti kerros Miniclienttien ja palvelimen väliin, missä olisi yksi asiakassovellus aina yhtä Miniclienttia kohtaan.

Miniclientilla voidaan testata pieni toimintokokonaisuus kerrallaan eikä se vaikuta



Kuva 3.2: Moduulikuva testiympäristöstä.

mittaustuloksiin. Samalla asiakaskoneella voi olla useita Miniclientteja suorittamassa testejä, jolloin voidaan keskittyä tutkimaan järjestelmän toimivuutta kuormassa. Miniclienttien lukumäärää ei ole rajattu, mutta niiden tehokkuus ja toimivuus riippuu asiakaskoneen asetuksista ja tehoista sekä simuloitavasta asiakassovelluksesta.

3.2.2 Toolbox

Toolbox on graafinen työkalu testien mallinnusta, tulosten ja monitoroitavien osien tietojen tarkastelua varten. Toolboxiin kehitettiin skenaariotyökalu, jolla voidaan määritellä, mitä testejä ja miten testejä suoritetaan.

Skenaariotyökalu

Skenaariotyökalu on kehitetty testien mallintamista varten. Sen avulla voidaan määrittää yksityiskohtaisesti, mitä ja miten testitapauksia suoritetaan. Skenaariotyökalussa voidaan luoda testijoukkoja.

Skenaariotyökaluun ei ole toteutettu graafiominaisuutta mallipohjaista testausta varten vähäisten resurssien takia. Graafeilla voisi mallintaa kätevästi järjestelmän käyttäytymistä ja käyttötapauksia kuvaavia malleja, jolloin testien ylläpidettävyys ja luettavuus paranisi. Kuitenkin sellaisen ominaisuuden lisääminen on suunnitelmissa ja siitä esitetään jatkokehitysajatuksia kohdassa 4.5.1.

Testitulosten tarkastelu

Toolboxissa on mahdollisuus lukea automaattisista testeistä saatuja tuloksia ja tutkia erilaisia kuvaajia, kuten testien vasteaikoihin liittyviä viivadiagrammeja. Testitulos tallentuu jokaisen testin suorituksen jälkeen automaattisesti tulostietokantaan, josta voidaan hakea kaikki testitulokset Toolboxille.

Tulosnäkymässä voidaan tutkia erikseen testijoukkojen ja -tapausten tuloksia sekä tarpeen vaatiessa yhdistellä tuloksia keskenään. Toolboxilla pystytään lukemaan yhteenvetoja testeistä sekä luomaan vertailtavia viiva- ja pylväskuvaajia eri testiversioiden välillä. Kaikki tulokset on mahdollista viedä CSV-taulukkotiedostoon ja kuvaajat voidaan tallentaa kuvatiedostoihin.

3.2.3 Testiesimerkkejä

Ennen testitulosten rakennetta tarkastellaan, miten erilaiset testit muodostuvat testipedissä. Tässä käydään läpi testijoukkojen muodostamisen perusideaa ja lopuksi esitellään esimerkkejä mallipohjaisista testijoukoista.

Hätäkeskustietojärjestelmässä tyypillinen päivystäjän toiminto on hätäilmoituksiin vastaaminen. Hätäilmoituksiin vastaamisesta on hyvä esittää esimerkki, miten siitä muodostetaan testijoukko. Ennen testijoukon muodostamista tutkitaan järjestelmän rakennetta ja etsitään yksittäiset potentiaaliset testitapaukset.

Taulukossa 3.1 on kuvattu yksittäisiä testitapauksia, joista voidaan tehdä erilaisia testijoukkoja sopivien testimallien ja käyttötapausten mukaisesti. Toiminnot-sarakkeessa on kuvattu, mitä toimintoja yksittäinen testitapaus sisältää ja mitä sisältyy vasteajan mittaamiseen. Esimerkiksi AnswerCall-testitapaus sisältää kolme toimintoa. AnswerCall-testitapauksen ensimmäistä toimintoa ei lasketa mukaan vasteaikaan, sillä ilmoitusten ilmentyminen käyttöliittymälle tapahtuu hätäilmoitussimulaattorin toimesta eikä voida olettaa, että ilmoituksia tulisi jatkuvasti. Joutenolon kestoa mitataan ja tallennetaan testitulokseen.

Taulukko 3.1: Esimerkkejä testitapauksista.

Testitapaus	Kuvaus	Toiminnot
StartClient	Käynnistää sovelluksen	1. Käynnistä sovellus. 2. Odota kirjautumisnäkymää.
Login	Kirjautuu järjestelmään	1. Valitse tehtävärooli. 2. Valitse hätäkeskusalue. 3. Kirjaudu sisään. 4. Odota päänäkymää.
AnswerCall	Vastaa ilmoitukseen	1. Odota ilmoitusta (joutenolo). 2. Vastaa hätäilmoitukseen. 3. Odota ilmoituslomaketta.
MarkAsPrankCall	Merkitse ilmoitus aiheettomaksi	1. Merkitse ilmoitus aiheettomaksi. 2. Odota toiminnon loppumista.
CloseIncidentForm	Sulje puhelu ja lomake	1. Sulje puhelu. 2. Sulje lomake. 3. Odota päänäkymää.

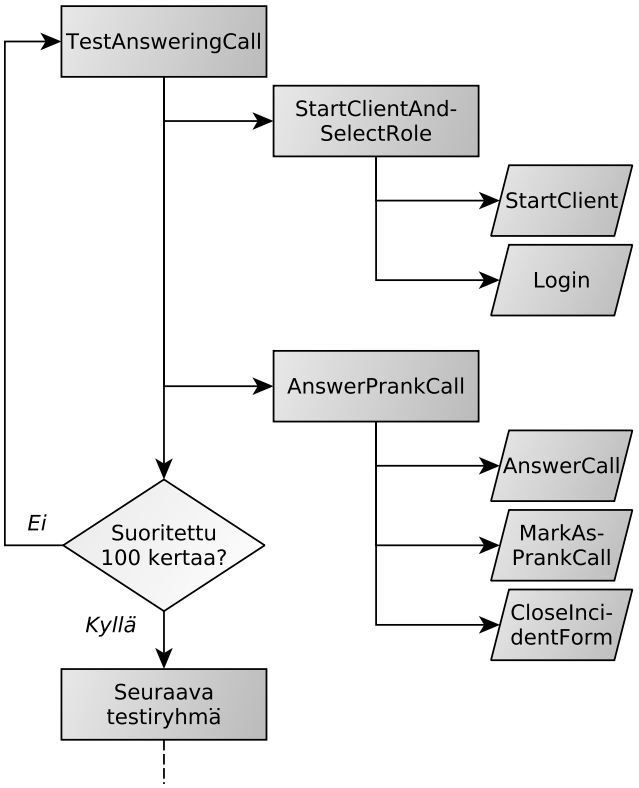
Aiemmin listatuista testitapauksista voidaan koostaa erilaisia testijoukkoja. Taulukossa 3.2 on esimerkkejä testijoukoista. Skenaarioiden rakentamisessa voidaan testijoukkoja yhdistellä muiden testijoukkojen kanssa. Koostaminen mahdollistaa sen, että testejä voidaan paloitella testaajan mieleiseksi, kuten TestAnsweringCall-testijoukko osoittaakin. Sarakkeessa Testit on kyseisen testijoukon sisältämät testit suoritusjärjestyksessä.

Taulukko 3.2: Esimerkkejä testijoukoista.

Testijoukko	Kierroksia	Testit
StartClientAndSelectRole	1	StartClient
		Login
AnswerPrankCall	1	AnswerCall
		MarkAsPrankCall
		CloseIncidentForm
TestAnsweringCall	100	StartClientAndSelectRole
		AnswerPrankCall

Kuvassa 3.3 on esimerkki testien puurakenteesta. Kuten kuvasta näkyy, testijoukot voivat olla muiden testijoukkojen sisällä, ja ne sisältävät testitapauksia. Tasaiset laatikot ovat testijoukkoja, vinosivuiset laatikot ovat ajettavia testitapauksia. Jokaisen testijoukon kierroksen jälkeen tarkastetaan, onko testijoukkoa ajettu määritellyn ajokierroksen verran — jos ei, lisätään kierrosmäärää yhdellä ja testijoukko ajetaan jälleen läpi. Kuvassa ei ole merkitty alitestijoukkojen kierrostarkastusta, mutta se tehdään joka tapauksessa.

Normaalien testijoukkojen kokoamisessa ei pystytä huomioimaan yhtä merkittävää ongelmaa testien suorituksessa: mitä tapahtuu, jos esimerkiksi ilmoitussimu-



Kuva 3.3: Esimerkki testijoukkojen puurakenteesta.

laattori ei luo uutta hätäilmoitusta? Testipeti jäisi odottamaan uutta hätäilmoitusta turhaan. Tällaisen tilanteen voi välttää käyttämällä normaalien testijoukkojen sijaan mallipohjaisia testijoukkoja.

Taulukossa 3.3 on esimerkkejä todennäköisyyteen perustuvista testijoukoista. Taulukossa on kaksi todennäköisyyteen perustuvaa testijoukkoa, jotka sisältävät samat testit. OpenOldIncident-testitapauksessa avataan vanha ilmoitus ja sille on annettu ensimmäisessä testijoukossa 20 % ja toisessa 65 % todennäköisyys. Testipeti valitsee suoritettavan testin todennäköisyyksiin perustuen ja arpomalla luvun 0-100 väliltä.

Taulukko 3.3: Esimerkkejä todennäköisyyteen perustuvista testijoukoista.

Testijoukko	Todennäköisyys	Testit
IncidentsOverflow	80 %	AnswerPrankCall
	20 %	OpenOldIncident
SmallNumberOfIncidents	35 %	AnswerPrankCall
	65 %	OpenOldIncident

Taulukossa 3.4 on esimerkki ehtoihin perustuvasta testijoukosta, jossa hyödynnetään taulukossa 3.3 esitettyjä testijoukkoja. Ehtoihin perustuvat testijoukot voivat sisältää yksittäisiä ehtoja tai moniehtoja sekä jokaista ehtoa kohden on määritelty jokin muu testijoukko tai -tapaus. Testipeti voi tutkia testattavan sovelluksen tilaa ja muuttujia, ja toteuttavien ehtojen mukaisesti valitsee sopivan testijoukon. Taulukossa 3.4 on esitetty ModelledTestGroup-testijoukko, joka sisältää yhden muuttujan sekä kaksi ehtoa ja testiä. Testipeti tutkii ilmoitusjonon kokoa, jonka perusteella testipeti valitsee suoritettavan testin.

Taulukko 3.4: Esimerkki ehtoihin perustuvasta testijoukosta.

Testijoukko	Ehto	Testit
ModelledTestGroup	Ilmoituksia < 2	SmallNumberOfIncidents
	Ilmoituksia ≥ 2	IncidentsOverflow

Esimerkiksi jos ilmoitusjonossa on vähemmän kuin kaksi ilmoitusta, testipeti valitsee SmallNumberOfIncidents-testijoukon suoritettavaksi. Koska valittu testijoukko perustuu todennäköisyyteen, testipeti arpoo luvun, jonka perusteella testipeti suoritetaan jompikumpi testeistä.

Monipuolisella testijoukkojen määrittämisellä voidaan antaa vapautta testaajalle, millaisia testejä suoritetaan. Uusia testitapauksia voidaan tehdä vanhojen rinnalle, päivittämällä sopivia testiryhmiä ja lisäämällä niille esimerkiksi suorittamisen todennäköisyydet.

3.3 Testipedin tuottama tulosaineisto

Testipeti tallentaa automaattisesti testien tuloksia jokaisen testin suorituksen jälkeen. Testipetiin voidaan määritellä käyttämään tietokoneen paikallista tai erillistä lähiverkon tulostietokantaa. Testausta voidaan skaalata tarpeen mukaan — pienten järjestelmien testauksessa voidaan hyödyntää paikallista tietokantaa. Suurten järjestelmien testauksessa voidaan klusteroida testaustoimintaa ja käyttää yhteistä tulostietokantaa.

Tulosten lisäksi tallennetaan monitoroitavien osien tietoja, kuten perustietoja testilaitteistosta, -ympäristöstä ja reaaliaikaista tietoa esimerkiksi järjestelmän muistinkulutuksesta.

3.3.1 Testitulosten rakenne

Tuloksiin kirjataan paljon erilaista tietoa, kuten testin onnistuminen, aikoja, nimiä ja mahdollisen virheen syy. Tietokantaan tallennetaan testin aloitus- ja lopetusajat, joiden erotuksesta saadaan testin vasteaika. Näiden lisäksi tallennetaan testin odotusaika, joka lasketaan testin aikana olevasta joutenolosta. Esimerkiksi testi voi jäädä odottamaan jonkin simulaattorin toimia, ennen kuin testi voi jatkaa. Tällöin lopullinen vasteaika tulee seuraavasta kaavasta:

$$Vasteaika = Lopetusaika - Aloitusaika - Odotusaika \tag{3.1}$$

Testitulokseen merkitään testin nimi sekä testattavan järjestelmän nimi, versio ja testin suorittaneen simulointisovelluksen nimi. Nimien avulla voidaan vertailla tuloksia eri testiversioiden välillä.

Epäonnistuneen testin tärkein tieto on testissä tapahtuneen virheen syy. Virheen syyn lisäksi edellisten tietojen, kuten vasteaikojen ja nimien, perusteella voidaan tutkia, miksi testi epäonnistui. Analysoinnilla pyritään jäljittämään virhe järjestelmästä tai testistä ja korjaamaan se.

3.3.2 Monitoroitavien osien tiedonkeräys

Testipeti voi monitoroida automaattisesti järjestelmän eri osia testauksen aikana ja tallentaa niiden tietoja tulostietokantaan. Esimerkiksi palvelimen tilaa voidaan tarkkailla jatkuvasti. Testipeti voi kerätä esimerkiksi järjestelmän perustietoja, kuten käyttöjärjestelmän tiedot, keskusmuistin määrä ja koko järjestelmän muistinkulutus.

Testipeti on kehitetty Java-ohjelmointikielellä, jonka tarjoaman rajapinnan kautta voidaan kerätä testattavan sovelluksen tietoja ajonaikaisesti. Esimerkiksi sovelluksen muistinkulutuksesta ja lukittuneiden säikeiden tiloista voidaan kerätä tietoa.

Testipedillä voidaan tällä hetkellä kerätä tietoa sekä asiakas- että palvelinsovelluksesta. Asiakassovelluksesta kerätään tietoa jokaisen testijoukon suorittamisen jälkeen ja palvelimesta tietoa kerätään minuutin välein. Palvelinta pyritään häiritsemään mahdollisimman vähän, ettei tiedon kerääminen aiheuta häiriötä tai viivettä testituloksiin.

Kerätyn tiedon avulla voidaan todentaa, että esimerkiksi aiemmin havaittu muistivuoto on saatu paikattua. Tietojen perusteella voidaan parantaa testattavan järjestelmän tehokkuutta optimoimalla eri toimintoja.

3.4 Testien määrittely

Testien määrittely on keskeinen osa ohjelmistojen testausta. Olennaisimmat kysymykset, mitä tulee testien määrittelyssä vastaan ovat: mitä testataan, miten testataan ja kuka määrittelee testit?

Koska korkean saatavuuden mittaaminen keskittyy järjestelmätestaustasolle, tulee testit määritellä toimitettavan järjestelmän näkökulmasta. Määrittelyssä pitää ottaa huomioon monia eri tekijöitä, kuten testattavan järjestelmän kypsyys kehityksen suhteen sekä muiden järjestelmään liittyvien komponenttien vaikutus testaamiseen. Testattavat versiot on valittava tarkkaan, koska testaus olisi turhaa, jos keskeisimpiä toimintoja ei ole vielä toteutettu järjestelmään.

Testauksessa täytyy huomioida testattavan järjestelmän eri testiversioiden erot, sillä testituloksia ei voi aina verrata keskenään. Versioiden väliset erot voivat olla esimerkiksi käyttäjätoimintojen muuttuminen radikaalisti tai testitapauksen muuttuminen.

3.4.1 Käytetyt testausmenetelmät

Luvussa 2 esiteltiin tässä työssä käytettävät testausmenetelmät, joita hyödynnetään korkean saatavuuden testaamisessa. Seuraavaksi käydään läpi tarkemmalla tasolla, miten voidaan mitata korkean saatavuuden laatutekijöitä luotettavasti.

Seuraavaksi määritellään käytettäviä testausmenetelmiä ja mahdollisia mittauspisteitä laatutekijöittäin. Ne ovat luotettavuus, skaalautuvuus, tehokkuus ja ylläpidettävyys. Luotettavuuteen sisällytetään saatavuuden mittaaminen.

Luotettavuus

Luotettavuutta määritteleviä asioita on paljon, joten pääpaino on luvussa 2 esitetyissä määreissä. Luotettavuutta määrittelevät osat ovat saatavuus, turvallisuus, ja varmuus järjestelmän toiminnasta, johon kuuluu myös virheensietokyky.

Testipedin ominaisuuksien avulla pyritään arvioimaan kyseiset luotettavuuden

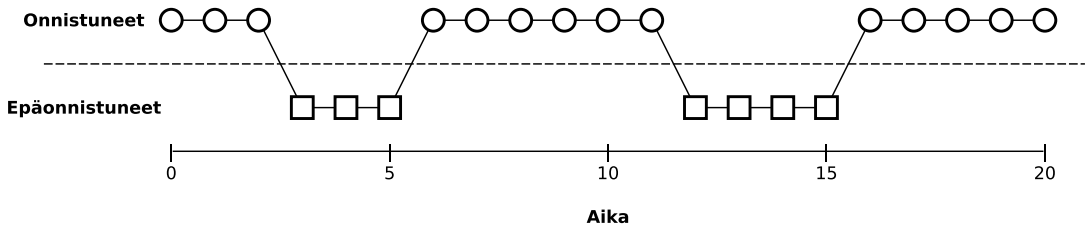
osat mahdollisimman tarkasti. Toimintavarmuus ja saatavuus arvioidaan testien suoritusmäärien avulla, eli kuinka paljon testejä järjestelmä läpäisee suhteessa testien kokonaismäärään. Turvallisuutta voidaan testata haavoittuvuustestauksen osalta ja testipedillä voidaan pyrkiä aiheuttamaan häiriötä järjestelmään. Virheensietokyky on järjestelmän piilevin ominaisuus, jota voidaan arvioida onnistuneiden testien määrän mukaan.

Saatavuus lasketaan järjestelmän toiminnassa oloajan mukaan. Schmidt esittää järjestelmän saatavuuden laskemiseksi seuraavan kaavan [4]:

$$Saatavuus = \frac{Toiminnassa}{Toiminnassa + Epäkunnossa} \tag{3.2}$$

Kaavassa 3.2 *Toiminnassa* tarkoittaa järjestelmän toimintakelpoisuusaikaa. *Epäkunnossa* tarkoittaa käyttökelvottoman järjestelmän aikaa.

Tässä työssä järjestelmän saatavuus pyritään laskemaan testitulosten perusteella, joten kaavaa 3.2 ei voi hyödyntää suoraan. Kuvassa 3.4 on esimerkki testipedin suorittamasta testikierroksesta. Onnistuneet testit on merkitty ympyröillä ja epäonnistuneet neliöillä. Suoritetut testit on muodostettu aikaleiman mukaiseen järjestykseen ja testien välissä oleva jana tarkoittaa testipedin suorittamaa testipolkua.



Kuva 3.4: Esimerkki suoritetusta testikierroksesta.

Testien suoritusten aikana järjestelmä voi häiriintyä hetkellisesti, kuten kuvan 3.4 esimerkistä voidaan nähdä. Järjestelmän saatavuus voidaan laskea onnistuneiden ja epäonnistuneiden testien kokonaiskeston mukaan. Testin kokonaiskestossa ei huomioida joutenoloaikaa, koska voidaan olettaa järjestelmän toimivan oletetulla tavalla myös joutenolon aikana. Muokkaamalla 3.2 kaavaa saadaan seuraavanlainen kaava saatavuuden mittaamiseen:

$$Saatavuus = \frac{\sum_{i=1}^{n}(Lopetus_i - Aloitus_i)}{Kaikkien\ testien\ kokonaiskesto} \tag{3.3}$$

Kaavassa 3.3 *Lopetus* tarkoittaa onnistuneen testin i lopetusaikaa, *Aloitus* vastaavasti testin i aloitusaikaa. Järjestelmän saatavuus voidaan laskea jakamalla onnistuneiden testien kokonaiskesto kaikkien testien kokonaiskestolla. Saatavuutta mitataan käyttäjän kokeman saatavuuden näkökulmasta.

Skaalautuvuus

Kuten luvussa 2 mainittiin, skaalautuvuutta on kahdentyyppistä: vertikaalista ja horisontaalista. Korkean saatavuuden mittauksessa on huomioitava molemmat skaalautuvuustyypit, jos niitä käytetään testattavassa järjestelmässä. Kuormittavilla testeillä voidaan testata järjestelmän kykyä säädellä toimintakykyään automaattisesti aiheuttamatta häiriötä.

Kuormittavien testien lisäksi voidaan tutkia myös eri tekniikoiden vaikutusta skaalautuvuuteen. Vertikaalista skaalautuvuutta voidaan tutkia vaihtamalla eri tietoteknisiä ratkaisuja muihin, ja testata muutosten vaikutusta. Tällainen muutos voi olla esimerkiksi tietokantajärjestelmän tai yhteysrajapinnan vaihtaminen toiseen ratkaisuun.

Horisontaalisen skaalautuvuuden tutkiminen on haastavampaa kuin vertikaalisen skaalautuvuuden. Järjestelmän pitäisi osata jakaa kuormaa muihin palveluihin tai tarvittaessa luoda uusia palveluita, jolloin täytyy tutkia järjestelmän käyttäytymistä suuren rasituksen aikana.

Suoritettujen testien lukumäärää voidaan suhteuttaa automaattisen testauksen kokonaisaikaan, josta voidaan nähdä, kuinka tehokas järjestelmä on tietyllä ajanhetkellä. Skaalautuvuuden vaikutusta voidaan mitata aluksi yhdellä testejä suorittavalla testipedillä, jonka tulosta voidaan verrata muihin tuloksiin, joissa on suoritettu useampia testejä yhtä aikaa. Samalla voidaan arvioida, osaako testattava järjestelmä tasapainottaa kuormaa ja kuinka hyvin.

Tehokkuus

Tehokkuus ja suorituskyky kertovat järjestelmän vasteesta suhteessa käyttäjän tekemiin toimintoihin. Tehokkuutta pyritään arvioimaan toimintojen vasteaikojen avulla. Testitulosten vasteaikoja hyödynnetään tehokkuuden arviointiin.

Graafisen käyttöliittymän sovelluksissa voi olla tyypillisesti useita operaatioita käyttäjän tekemän toiminnon takana, jolloin on otettava huomioon käyttäjän kokema vasteaika. Testipedin avulla testien vasteajat voidaan mitata toiminnon alkamisesta aina järjestelmän vasteeseen asti. Testaajan vastuulla on määritellä sellaisia testitapauksia, jotka mittaavat yhden toiminnon vasteaikaa sekä mahdollisesti jäävät odottamaan järjestelmästä saatavaa vastetta.

Tehokkuutta voidaan tutkia myös luotettavuuden ja skaalautuvuuden osalta. Esimerkiksi tiukat tietoturvatekniikat voivat heikentää järjestelmän tehokkuutta, mutta nostaa luotettavuutta. Skaalautuvuus pyrkii parantamaan tehokkuutta.

Ylläpidettävyys

Ylläpidettävyyttä ei voi suoraan tarkastella automaattisen testauksen kanssa, vaan se pitää tehdä automaattisen testauksen ulkopuolelta. Ylläpidettävyyttä voitaisiin arvioida virheiden korjausten perusteella. Jos testituloksista löydetään selkeitä syitä epäonnistuneisiin testeihin, niin voidaan arvioida, kuinka helposti vika olisi korjattavissa. Arviointiin ei voida täysin luottaa ylläpidettävyyttä mitattaessa, koska arviointi perustuisi kehittäjän subjektiiviseen näkemykseen.

Ylläpidettävyyttä voidaan mitata eri metriikoiden ja analysointityökalujen avulla. Esimerkiksi koodin kompleksisuutta voidaan mitata eräillä analysointityökaluilla. Ylläpidettävyyttä voidaan edistää esimerkiksi vakiinnuttamalla eri menettelytapoja ja standardeja koko kehitysprojektiin. Eräitä ylläpidettävyyttä parantavia menettelytapoja ovat koodin ja testauksen katselmoinnit.

3.4.2 Testien määrittelijät

Automaattisen testauksen ja etenkin testien määrittelyt tulee tehdä mahdollisimman varhaisessa vaiheessa kehitysprojektissa. Testauksen määrittelyssä haasteena on, kuka tekee testit ja mistä testit voidaan määritellä.

Tyypillisesti kehitysprojektissa tehdään kattava määrittely koko projektista. Testaajalle tärkein dokumentti on määrittelydokumentti, koska testaus perustuu asiakkaalle luvattuun toteutukseen ja järjestelmään kehitettäviin ominaisuuksiin. Ketterien kehitysprojektien ongelma on, ettei testaajalla ole aina mahdollista saada tarkkaa kuvausta kehitettävästä järjestelmästä, koska sen määrittely tarkentuvat ajan myötä. Testaaja ei silloin kykene muodostamaan kunnollisia testitapauksia hajanaisen dokumentaation perusteella, jolloin testitapaukset muodostetaan muun tiedonlähteen avulla.

Seuraavaksi testaajalle tärkeitä tiedonlähteitä ovat käyttäjä ja asiakas sekä ketterissä kehitysprojekteissa järjestelmän tuoteomistajat. Ketterissä kehitysprojekteissa asiakasta pyritään pitämään mahdollisimman lähellä kehitystä, jolloin voidaan määritellä kehitettävää järjestelmää joustavasti. Asiakas antaa tietoa siitä, mitä järjestelmältä odotetaan ja mitkä ovat sen tyypilliset käyttötapaukset. Käyttäjä voi antaa käyttötavoistaan tietoa, jota voidaan hyödyntää sekä kehityksessä että testauksessa. Näiden tietojen perusteella voidaan muodostaa erilaisia käyttötapauksia ja niiden pohjalta edelleen testitapauksia. Asiakas tai käyttäjä voivat myös määritellä hyväksyttäviä suorituskyky- ja luotettavuuskriteereitä.

Testien määrittely on järjestelmän toimittajan ja varsinkin testaajien vastuulla. Käyttäjien tuoman tiedon lisäksi voidaan hyödyntää kehittäjien tietoa eri toimintojen testaamiseen. Eri tietojen avulla testaaja voi määritellä sellaisia testitapauksia, jotka antavat tärkeää tietoa järjestelmän laadusta.

4. TYÖN TULOKSET

Automaattisen testauksen kehittäminen ei ole suoraviivaista, vaan se vaatii paljon aikaa ja resursseja. Testaustyökalun kehittämisessä tulee ottaa huomioon, että yleiskäyttöisen työkalun kehittäminen voi olla vaativampaa kuin projektikohtaisen työkalun.

Testipedin kehityksessä kohdatut haasteet ja hyödyt pyritään tuomaan esille tässä luvussa. Kehityksessä saatuja kokemuksia pyritään vertaamaan luvussa 3 esitettyjä haasteita ja hyötyjä vasten. Testipedin kehitysvaiheen kuvauksen jälkeen esitetään testeistä saadut tulokset.

4.1 Testipedin kehitysvaihe

Ennen automaattiseen testaukseen perustuvan työkalun kehitystä tulisi punnita erilaisia vaihtoehtoja, jotka voisivat tarjota testaukselle samat ominaisuudet kuin kehitettävä työkalu. Testipetiä aloitettiin kehittämään, koska korkean saatavuuden testaamiseen ei löytynyt sopivaa työkalua.

Testipedin kehittämisen aikana suurimmaksi haasteeksi muodostui resursointi. Testipetiin suunniteltuja ominaisuuksia jouduttiin priorisoimaan muun muassa kehittäjämäärän ja aikataulun tuottaman paineen takia. Kehityksessä otettiin huomioon vain tärkeimmät ominaisuudet, jotka tuli kehittää ennen muiden ominaisuuksien tekemistä.

Testipedin kehittämisestä oli paljon hyötyä, koska saatiin työkalu, jonka avulla voitiin arvioida järjestelmän korkeaa saatavuutta testitulosten avulla eri laatutekijöiden näkökulmasta. Suurin aika meni kehityksen lisäksi testitulosten analysointiin ja raportointiin.

Kehityksen varhaisessa vaiheessa pystyttiin testaamaan järjestelmää ja näkemään testipedin tuoma hyöty, koska sitä kehitettiin iteratiivisesti. Kehityksen aikana pystyttiin keksimään, mitä hyödyllisiä ominaisuuksia voitaisiin kehittää. Korkean saatavuuden testipetiä kehitetään edelleen ja testipedin kehitykseen liittyviä jatkokehitysajatuksia on esitetty luvussa 4.5.

4.2 Testipedin varhainen versio

Testipedin esiversiolla mitattiin eri toimintojen vasteaikoja asiakassovelluksen käyttöliittymätasolla. Manuaalisessa testauksessa havaittiin, että asiakassovelluksen

käyttöliittymä vastasi erittäin hitaasti käyttäjän syötteisiin. Tämä tilanne havaittiin hyväksi mahdollisuudeksi testata testipedin toimivuutta. Testauksessa edettiin seuraavissa työvaiheissa:

1. Testeihin sopivien rajapintojen etsintä testattavasta sovelluksesta.
2. Testitapausten kirjoittaminen.
3. Testijoukkojen määrittely.
4. Testien suorittaminen testipedissä.
5. Asiakassovelluksen muokkaaminen ja testien toistaminen tarpeen vaatiessa.
6. Tulosten ja asiakassovelluksesta löytyneiden virheiden raportointi.

Aluksi tutkittiin, mitkä toiminnot olisivat potentiaalisia testitapauksia hitauden löytämiseksi ja mitä rajapintoja voitiin hyödyntää testauksessa. Tämä vaati asiakassovelluksen koodin tutkimista, jossa tarkasteltiin mahdollisia hitautta aiheuttaneet toimintoja. Lisätietoa saatiin tarvittaessa manuaalisen testauksen suorittaneilta testaajilta.

Kun hitautta aiheuttaneet toiminnot löydettiin, koodista etsittiin oikeat luokat ja niiden tarjoamia rajapintoja hyödynnettiin testitapausten kehittämisessä. Asiakassovelluksen omien rajapintojen lisäksi pystyttiin hyödyntämään myös sen käyttämää graafista käyttöliittymäkirjastoa esimerkiksi painikekomponentin aktivoimiseen.

Hitautta aiheuttanut toiminto liittyi keskeneräisten ilmoitusten valintaan. Toiminnon testausta varten määriteltiin testijoukko. Testijoukko sisälsi seuraavat testit:

1. Käynnistä sovellus.
2. Kirjaudu järjestelmään.
3. Vaihda valintaa keskeneräisten tehtävien listassa 100 kertaa.

Testipedin varhaisversio tallensi testitulokset XML-tiedostoihin ja vasteaikojen vertailu tehtiin manuaalisesti. Testien jälkeen asiakassovelluksesta karsittiin eri toimintoja pois, millä pyrittiin etsimään hitautta aiheuttaneet kohdat. Toimintojen karsimisen jälkeen sovellusta testattiin uudelleen.

Testauksessa kävi ilmi, että asiakassovellukseen ladattiin ajonaikaisesti eri käyttöliittymätyylejä jokaisen keskeneräisen ilmoituksen valinnan jälkeen ja samanlaisia tyylejä saattoi olla ladattuna useampaan kertaan. Tyylien turha lataaminen aiheutti asiakassovelluksessa graafista laskentaa jatkuvasti ja se hidasti sovelluksen toimintaa.

Tässä huomattiin, että asiakassovelluksen muokkaaminen ja testien toistaminen on tärkeä osa automaattista testausta, koska sillä säästetään selvästi aikaa manuaaliseen testaukseen verrattuna. Varhaisversiolla testaaminen oli osittain myös testipedin testaamista, toimiiko se niin kuin halutaan.

4.3 Varotietopalvelun testaus

Testipedin seuraavaan versioon oli kehitetty Miniclient- ja Test Agent -toteutukset. Tuloksia pystyi tutkimaan myös suoraan Toolboxista. Testipeti tallensi testitulokset tulostietokantaan automaattisesti, eikä XML-tiedostoja käytetty tulosten keräämiseen. Edellä mainittujen ominaisuuksien lisäksi testipedin tulokset pystyttiin tallentamaan CSV-tiedostoon raportointia varten.

Ensimmäisenä testikohteena oli Varotietopalvelu, jota pyrittiin testaamaan luotettavuuden ja tehokkuuden näkökulmista. Testausta varten tehtiin Miniclient-toteutus, joka hyödynsi oikean asiakassovelluksen rajapintoja.

4.3.1 Ensimmäisen vaiheen testitapaukset

Testeihin sisällytettiin seuraavat testattavat toiminnot: varotietojen lisääminen, muokkaaminen ja poistaminen. Taulukossa 4.1 on kuvattu testijärjestys sekä kaikki käytetyt testitapaukset. Testipeti pyrki suorittamaan kaikki CautionInformation-testijoukon sisältämät testit välittämättä siitä, jos jokin testitapaus epäonnistuu.

Taulukko 4.1: Varotietopalvelun testit suoritusjärjestyksessä.

Testijoukko	**Testitapaus**	**Kuvaus**
CautionInformation	CreatePerson	Luo henkilövarotiedon
	CreateAddress	Luo osoitevarotiedon
	CreateVehicle	Luo ajoneuvovarotiedon
	ModifyPerson	Muokkaa henkilövarotietoa
	ModifyAddress	Muokkaa osoitevarotietoa
	ModifyVehicle	Muokkaa ajoneuvovarotietoa
	RemoveAddress	Poistaa osoitevarotiedon
	RemoveVehicle	Poistaa ajoneuvovarotiedon
	RemovePerson	Poistaa henkilövarotiedon

Testeissä oli huomioitava tarkka testijärjestys, koska palvelimen tietokanta oli lähtötilanteessa aina tyhjä. Lisäykseen liittyvät testitapaukset suoritettiin aina ennen muokkaus- ja poistotestejä. Viimeinen poistotesti oli väistämättä henkilövarotiedon poisto, koska jokainen osoite- ja ajoneuvovarotieto on aina sidonnainen yhteen tai useampaan henkilövarotietoon.

4.3.2 Ensimmäisen vaiheen testitulokset

Varotietopalvelua testattiin edellä mainitulla testijoukolla neljä kertaa. Testaukset suoritettiin kehitysversioilla, joten testitulokset esitellään Varotietopalvelun versionumeroiden mukaan.

Ensimmäinen Varotietopalvelun testattu versio oli 2.2.3, johon ei huomioitu aiempia muutoksia. Kaikki muutokset esitellään aina verraten edelliseen testattuun versioon, jotta tuloksia voidaan tarkastella objektiivisesti ja ottaa kantaa muutosten vaikutuksesta tuloksiin. Kaikki tarkemmat Varotietopalvelun testitulokset ovat liitteessä 1.

Varotietopalvelun ensimmäisessä testatussa versiossa oli käytetty tietokantatekniikkana H2:sta ja yhteystekniikkana RMI:tä. Testipeti suoritti versiossa 2.2.3 kaikkia testitapauksia lähes 3250 kertaa, joista suurin osa oli onnistuneita testejä.

Versiossa 2.2.3 osoitevarotiedon poistaminen epäonnistui 124 kertaa, ja ajoneuvovarotiedon poistaminen epäonnistui 438 kertaa. Testijoukon suorittaminen kokonaisuudessaan epäonnistui 443 kertaa. CautionInformation-testijoukon onnistumisprosentti oli 86 %.

Muokkaustoimintojen testitapaukset olivat vasteajoiltaan kohtuullisia — tietojen muokkaukset kestivät keskiarvoiltaan 77-109 millisekuntia ja mediaaneiltaan 63-94 millisekuntia. Lisäys- ja poistotoiminnot olivat vasteajoiltaan erittäin hitaita, esimerkiksi henkilövarotiedon lisääminen kesti keskiarvoltaan 3,7 sekuntia ja tyypillisesti lähes 1,5 sekuntia.

Versiossa 2.2.3 osoite- ja ajoneuvovarotiedon lisäystoiminnot olivat huomattavasti hitaampia kuin henkilövarotiedon lisäystoiminto. Mediaaneiltaan vasteajat olivat osoitevarotiedon lisäyksessä 2917 millisekuntia, ajoneuvovarotiedon lisäyksessä 2652 millisekuntia ja henkilövarotiedon lisäyksessä 1482 millisekuntia. Hitaus kyseisissä lisäystoiminnoissa johtui siitä, että molemmat toiminnot vaativat aina uuden henkilövarotiedon lisäämisen.

Varotietopalvelua testattiin uudestaan versiossa 2.2.4. Versiossa 2.2.4 oli optimoitu olioiden tallentamista tietokantaan, jolloin oliot veivät vähemmän tilaa tietokannasta.

Varotietopalvelun luotettavuus parani versiossa 2.2.4 verrattuna versioon 2.2.3, koska suoritettujen testien määrä kasvoi ja CautionInformation-testijoukon onnistumisprosentti on 90 %. Testitapauksia suoritettiin kaikkiaan 3149 kertaa, ja koko testijoukon suorituksista epäonnistui 311 kappaletta.

Versiossa 2.2.4 testien vasteajat paranivat, koska tietokantaan tallennettavien olioiden tiedon määrää karsittiin. CautionInformation-testijoukon vasteaika parani keskiarvoltaan noin 2,5 sekuntia ja mediaaniltaan lähes 700 millisekuntia.

Seuraava testattu Varotietopalvelun versio oli 2.2.5, jossa oli korjattu edellisessä

versiossa löydetyt virheet. Tietokantaan tallennuksen lisäksi osoite- ja ajoneuvovarotietojen poistotoiminnot oli korjattu.

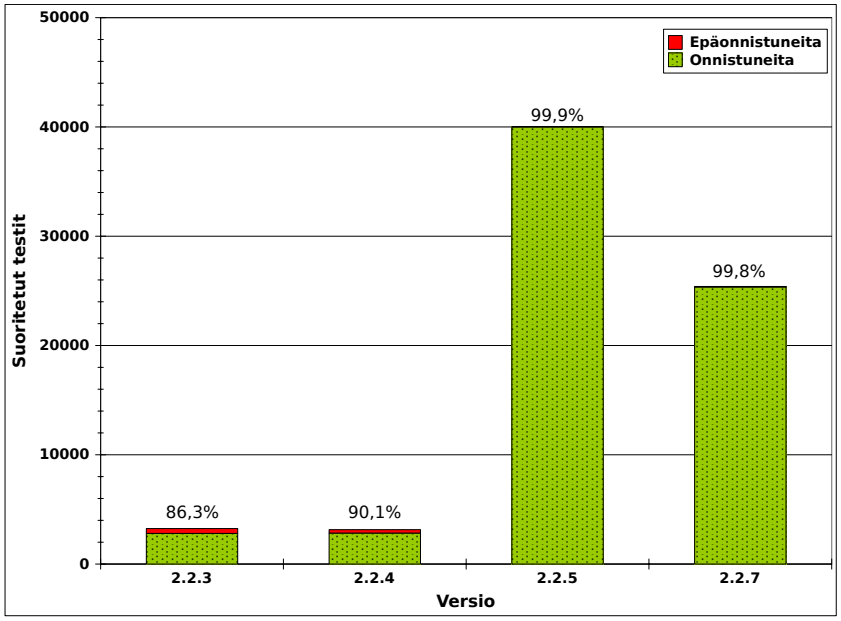
Versiossa 2.2.5 järjestelmän tehokkuus parani selvästi ja testimäärät nousivat rajusti verrattuna edellisiin versioihin. CautionInformation-testijoukkoa suoritettiin 40031 kertaa. Testien onnistumismäärän kasvamisen myötä testien vasteajatkin paranivat selkeästi — koko testijoukon suorittamiseen meni keskimäärin 4,3 sekuntia ja mediaaniltaan 2,2 sekuntia. Testien onnistumisprosentti oli pienimmillään 99,9 %.

Versiossa 2.2.5 lisäystoimintojen vasteajat paranivat huomattavasti, esimerkiksi osoitevarotiedon lisäys kesti keskiarvoltaan 935 millisekuntia ja mediaaniltaan 447 millisekuntia. Edellisiin versioihin verrattuna lähes kaikki vasteajat parantuivat, mutta varohenkilön poistotoiminnon osalta tehokkuus heikkeni: keskiarvoltaan se kesti 900 millisekuntia ja mediaaniltaan 282 millisekuntia.

Seuraava testattu Varotietopalvelun versio oli 2.2.7, jossa oli muutettu tietokantajärjestelmä H2:sta PostgreSQL:ään ja yhteystekniikka RMI:stä JMS:ään. Versiossa 2.2.7 testejä suoritettiin vähemmän, koska testipeti oli jumiutunut testauksen aikana. Testausta jatkettiin heti, kun ongelma havaittiin ja korjattiin. Luotettavuus oli samaa luokkaa edellisten tulosten kanssa — testejä suoritettiin noin 25400 kertaa ja testijoukon onnistumisprosentti oli 99 %.

Version 2.2.5 ja 2.2.7 välillä tehdyt tekniikkamuutokset vaikuttivat erittäin vähän testien vasteaikoihin. Osa vasteajoista heikkeni ja osa parani verraten edellisiin testaustuloksiin. Vasteajoiltaan keskimääräiset ja mediaanit olivat lähes samanlaisia, mutta minimivasteajat kasvoivat versiossa 2.2.7 korkeintaan kymmenillä millisekunneilla.

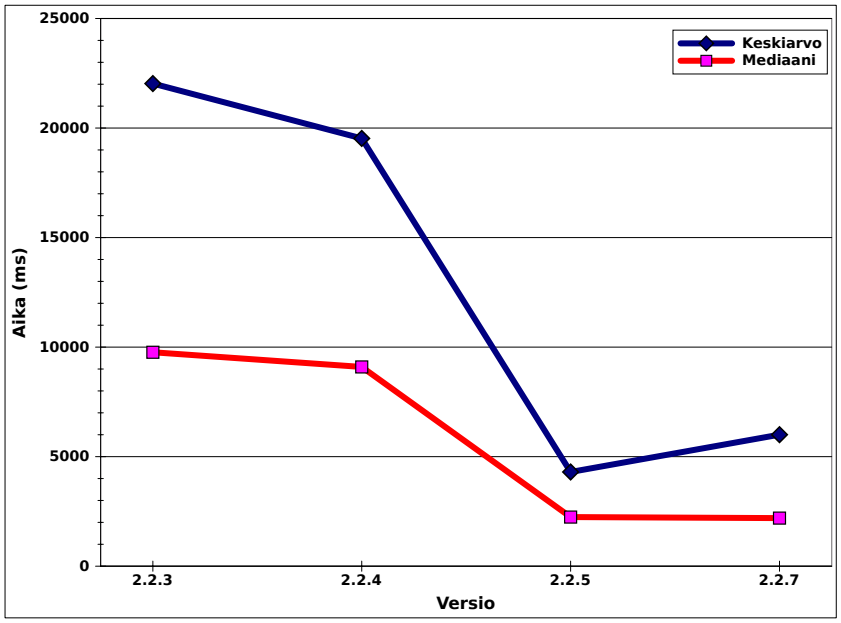
Kuvassa 4.1 esitetään CautionInformation-testijoukon onnistuneiden ja epäonnistuneiden suorituksien määrää. Jokaisen pylvään päällä olevat prosentit kertovat onnistuneiden testien osuuden kaikista suorituksista. Kuvasta voidaan nähdä, että Varotietopalvelun tehokkuus ja luotettavuus olivat huomattavasti parantuneet. Kokonaisuudessaan CautionInformation-testijoukon suoritusmäärä nousi lähes 37000 suorituksella versioiden 2.2.4 ja 2.2.5 välillä. Varotietopalvelun versioissa 2.2.5 ja 2.2.7 testijoukon epäonnistumisia oli erittäin vähän.



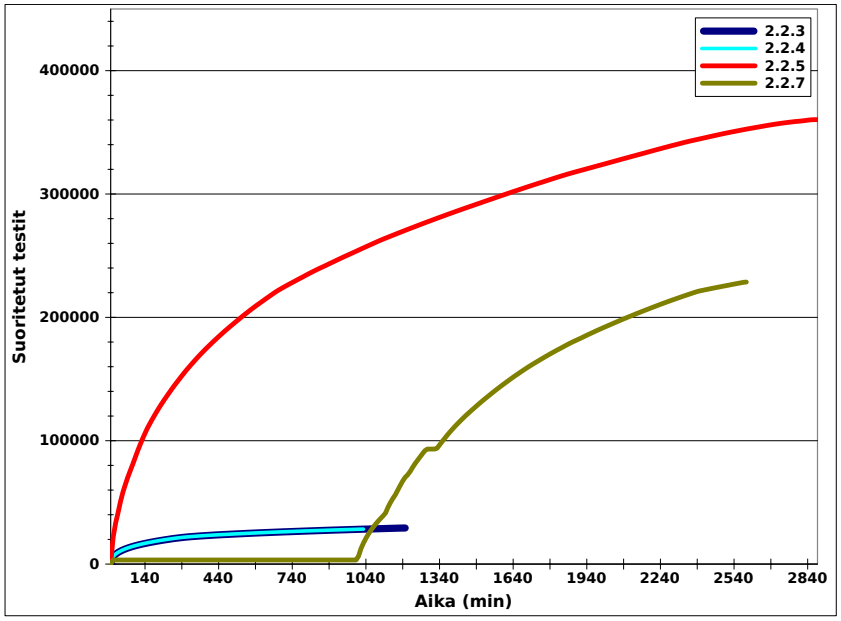
Kuva 4.1: CautionInformation-testijoukon suoritusmäärät.

Kuvassa 4.2 on esitetty CautionInformation-testijoukon vasteajat versioittain. Vasteajoista on otettu huomioon vain keskimääräinen ja mediaani. Kuvasta voidaan tulkita, että vasteajat ovat selkeästi parantuneet versioiden 2.2.4 ja 2.2.5 välillä. Koko testijoukon suoritukseen oli keskimääräisesti mennyt pienimmillään lähes viisi sekuntia, missä olisi parannettavan varaa. Kuitenkin testijoukon suorittamista hidastaa ajoneuvo- ja osoitevarotiedon lisäystestien yhteydessä tehtävät uudet henkilövarotiedot.

Kuvassa 4.3 on esitetty kaikkien suoritettujen testien määrää suhteessa kokonaissuoritusaikaan. Kuvasta voidaan nähdä, että Varotietopalvelu hidastuu ajan kulues-



Kuva 4.2: CautionInformation-testijoukon vasteajat.



Kuva 4.3: Testien suoritusmäärät suhteessa aikaan.

sa. Järjestelmän tehottomuutta ilmentää versioiden 2.2.3 ja 2.2.4 kuvaajat, joista nähdään, että järjestelmä kykeni toimimaan noin 17 tunnin ajan.

Kuvassa 4.3 versioiden 2.2.5 ja 2.2.7 kuvaajat muistuttavat muodoiltaan toisiaan. Tästä voidaan päätellä, että toiminnot ovat yhtä tehokkaita, vaikka versiossa 2.2.7 tietokantajärjestelmä ja yhteystekniikka vaihdettiin. Version 2.2.7 testauksen aikana testipeti jumiutui, ja se huomattiin noin 17 tuntia myöhemmin, kuten kuvasta näkyy. Testipeti alustettiin ja testejä jatkettiin jumiutumisen jälkeen.

4.3.3 Toisen vaiheen testitapaukset

Ensimmäisessä vaiheessa testeistä löydettiin oleellinen virhe: tietokantaan lisätään turha henkilövarotieto aina osoite- tai ajoneuvovarotiedon lisäyksen yhteydessä. Tällöin jokaisen testijoukon kierroksen jälkeen tietokantaan jäi kaksi henkilövarotietoa, vaikka kaikki varotiedot olisi pitänyt poistaa poistotestien aikana. Testitapaukset korjattiin Varotietopalvelun toiseen vaiheen testaukseen ja uudet testitapaukset suoritettiin Varotietopalvelun versioiden 2.2.20 ja 2.2.21 kohdalla. Taulukossa 4.2 on esitelty uudet testitapaukset.

Taulukko 4.2: Varotietopalvelun uudet testit suoritusjärjestyksessä.

Testijoukko	Testitapaus	Kuvaus
NewCautionInformation	CreateAllInformation	Luo henkilö-, osoite- ja ajoneuvovarotiedon
	ModifyPerson	Muokkaa henkilövarotietoa
	ModifyAddress	Muokkaa osoitevarotietoa
	ModifyVehicle	Muokkaa ajoneuvovarotietoa
	RemoveAddress	Poistaa osoitevarotiedon
	RemoveVehicle	Poistaa ajoneuvovarotiedon
	RemovePerson	Poistaa henkilövarotiedon

CreateAllInformation-testitapaus suorittaa kaikkien varotietojen lisäyksen. Jokaiseen luotuun henkilövarotietoon liitetään osoite- ja ajoneuvovarotieto. Testijoukon suorittamisen jälkeen tietokantaan ei jää yhtään varotietoa talteen.

4.3.4 Toisen vaiheen testitulokset

Varotietopalvelua testattiin kahdesti uusilla testeillä. Ensimmäisen vaiheen tavoin, testaukset suoritettiin kehitysversioilla, joten testitulokset esitellään Varotietopalvelun versionumeroiden mukaan.

Toisen vaiheen ensimmäinen testattu versio oli 2.2.20, jota verrataan edellisiin testituloksiin karkealla tasolla, koska testejä muutettiin. Varotietopalvelun toisen vaiheen testitulokset ovat tarkemmin esitelty liitteessä 1.

Testatussa versiossa oli käytetty tietokantajärjestelmänä PostgreSQL:ää ja yhteystekniikkana JMS:ää. Verrattuna edelliseen versioon, tähän versioon tehtiin paljon korjauksia sekä uusia ominaisuuksia.

Testipedillä saatiin suoritettua enemmän testejä, mutta järjestelmän luotettavuus heikkeni, kuten tuloksista huomataan. Testejä suoritettiin yli 58000 kertaa ja onnistuneiden testien osuus on 98 %.

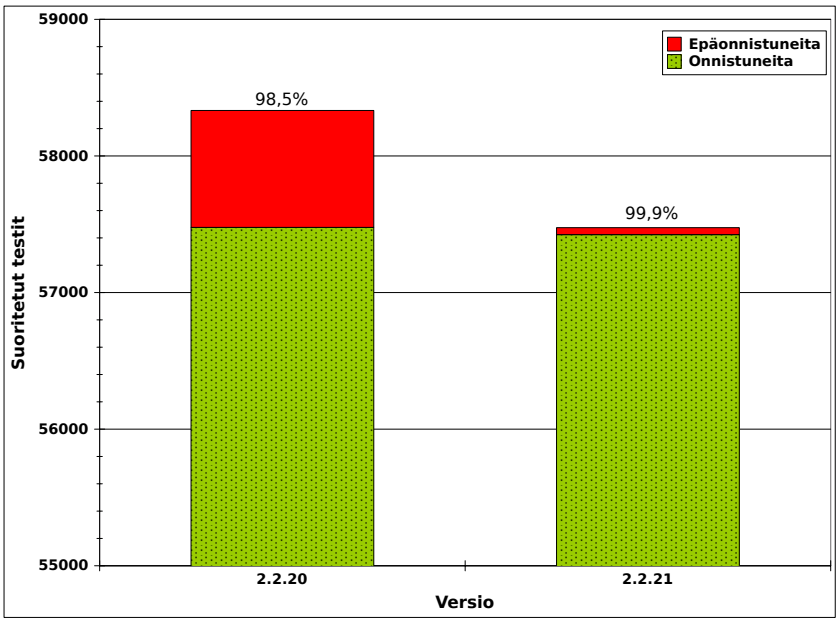
Verrattuna Varotietopalvelun version 2.2.7 tuloksiin, testitapausten vasteajat paranivat huomattavasti. Muokkaus- ja poistotoimintojen mediaanivasteajat olivat korkeintaan 30 millisekuntia ja kaikkien varotietojen lisäys kesti samassa testissä mediaaniltaan 320 millisekuntia.

Varotietopalvelun versiossa 2.2.21 käytetyt tekniikat eivät muuttuneet. Verrattuna edellisen version testituloksiin, Varotietopalvelun luotettavuus parani korjausten ansiosta. Testimäärissä on nähtävissä samankaltaisuutta verrattuna version 2.2.20 tuloksiin. Onnistuneiden testien osuus oli 99,9 %.

Testien keskimääräiset vasteajat paranivat ja testit olivat mediaaneiltaan yhtä tehokkaita kuin aiemmin. Keskimääräiset vasteajat paranivat, koska versiossa 2.2.20 oli enemmän epäonnistuneita testejä, joiden vasteajat olivat suuria ja ne vaikuttivat

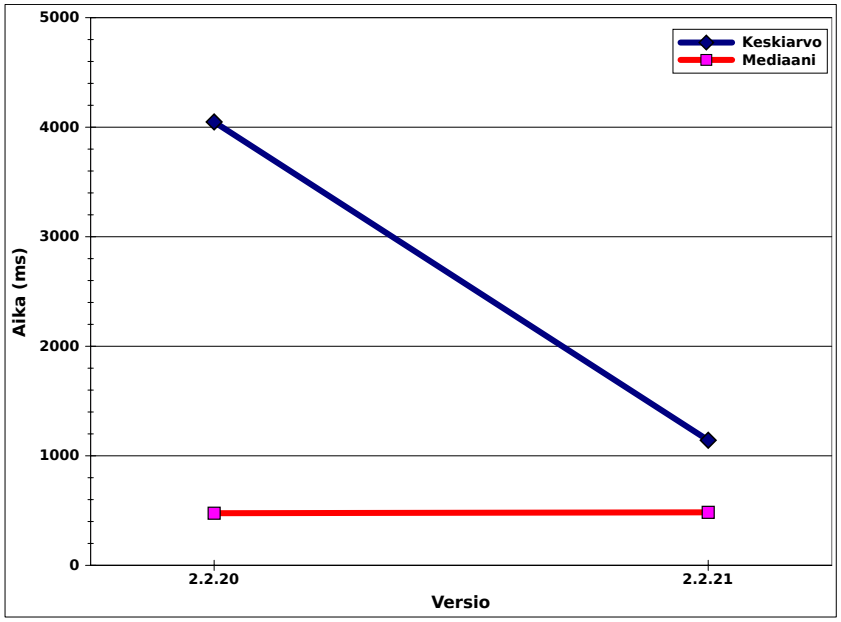
keskimääräiseen vasteaikaan huomattavasti.

Kuvassa 4.4 on esitetty NewCautionInformation-testijoukon onnistuneiden ja epäonnistuneiden suorituksien lukumäärää. Molempien pylväiden päällä olevat prosentit kertovat onnistuneiden testien osuutta kaikista suorituksista. Kuvasta voidaan huomata, että versiossa 2.2.20 oli enemmän epäonnistuneita testejä kuin versiossa 2.2.21. Kuva vääristää epäonnistuneiden ja onnistuneiden testien välistä suhdetta, koska kuvaajan alaraja alkaa 55000 suorituksesta. Molemmissa versioissa onnistuneita testejä oli suoritettu saman verran eli noin 57400 kappaletta.



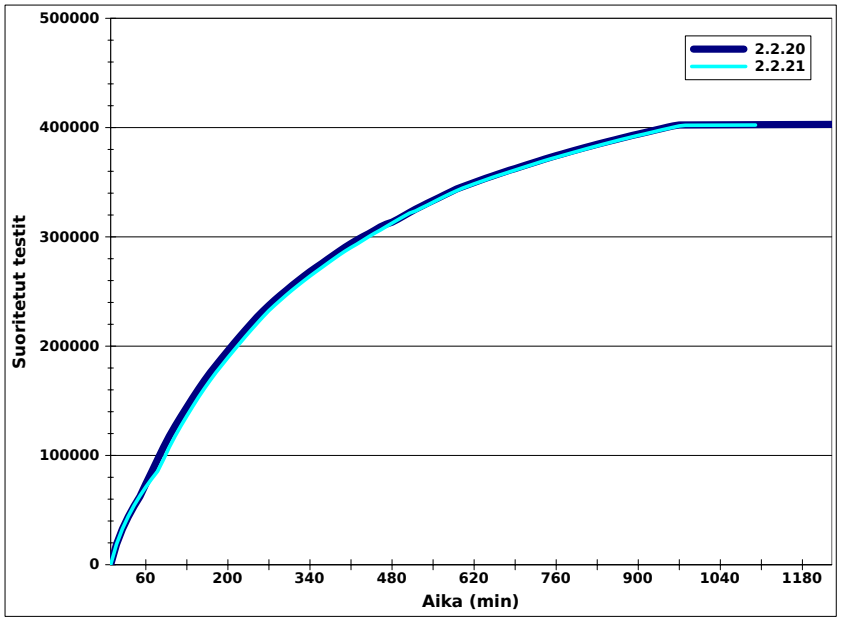
Kuva 4.4: NewCautionInformation -testijoukon suoritusmäärät.

Kuvassa 4.5 on esitetty NewCautionInformation-testijoukon vasteajat molemmista testiversioista. Vasteajoista on otettu huomioon vain keskimääräinen ja mediaani. Kuvasta voidaan nähdä, että NewCautionInformation -testijoukon keskimääräinen vasteaika oli huomattavasti pienempi versiossa 2.2.21. Tämä johtuu siitä, että versiossa 2.2.21 oli vähemmän epäonnistuneita testejä, jolloin testijoukon vasteaika oli pienempi kuin edeltävässä versiossa.



Kuva 4.5: NewCautionInformation -testijoukon vasteajat.

Kuvassa 4.6 kuvataan kaikkien ajettujen testien kokonaissuoritukset suhteessa testipedin ajoaikaan. Kyseisestä kuvasta on rajattu pois version 2.2.20 tulosviivan loppuosa, koska se jatkuisi tasaisena aina 3940 minuuttiin asti. Kuvasta voidaan nähdä, että molemmat Varotietopalvelun versiot olivat yhtä tehokkaita. Käyrien jyrkkyydestä voidaan päätellä, että Varotietopalvelu on erittäin tehokas käsittelemään suurta tietomäärää lyhyessä ajassa. Tämä ei vastaa kuitenkaan realistista käyttötapausta, jossa Varotietopalvelussa olisi muutamia aktiivisia käyttäjiä, jolloin järjestelmään kohdistuisi huomattavasti pienempää kuormaa kuin mitä testipeti kykenee aiheuttamaan.



Kuva 4.6: Uusien testien suoritusmäärät suhteessa aikaan.

Kuvasta 4.6 huomataan, että kummankin version tehokkuus romahtaa 400 000 testin suorituksen kohdalla. Romahdus kertoo siitä, että testit alkoivat epäonnistua ja Varotietopalvelu ei enää kyennyt toimimaan luotettavasti. Tämä johtui tietokannan täyttymisestä äärimmilleen, jolloin järjestelmän toiminnot alkoivat hidastumaan huomattavasti ja testit epäonnistuivat annetun aikarajan takia. Automaattisen testauksen hyöty on nähtävissä sen nopeudessa, koska jo tuhansien testien suorittaminen korkealla suoritustarkkuudella olisi vaatinut manuaaliselta testaukselta paljon aikaa.

4.3.5 Yhteenveto

Varotietopalvelun korkean saatavuuden testaus osoittautui suoraviivaiseksi prosessiksi. Prosessi eteni testien suunnittelusta suoritukseen ja testitulosten tulkinnasta arviointiin.

Testauksen aikana löydettiin selkeitä virheitä ja tehottomia toimintoja testatusta järjestelmästä. Virheelliset toiminnot korjattiin ja tietokantaan liittyvät optimoinnit tehtiin tulosten perusteella. Varotietopalvelun luotettavuutta ja tehokkuutta saatiin parannettua huomattavasti.

Varotietopalvelun versioiden 2.2.5 ja 2.2.7 välillä vaihdettiin tietokantajärjestelmä H2:sta PostgreSQL:ään ja yhteystekniikka RMI:stä JMS:ään. Tekniikkojen muutokset eivät muuttaneet tuloksia millään tavalla.

Testit saatiin suoritettua ja testipeti kokosi selkeät testitulokset. Varotietopalvelun ensimmäisessä testausvaiheessa havaittiin virhe lisäystoimintojen testitapauksissa. Virhe korjattiin toisen testausvaiheen testitapauksiin, jolloin yhdistettiin lisäystoimintojen testitapaukset yhdeksi testitapaukseksi.

4.4 Hätäkeskustietojärjestelmän testaus

Testaus suoritettiin kahdella eri järjestelmäversiolla: 2.5.1 ja 2.5.4. Molemmissa versioissa testattiin perinteisen automaattisen testauksen ja mallipohjaisen testauksen toimivuutta tätä työtä varten. Taulukossa 4.3 on kuvattu kaikki testauksessa käytetyt testitapaukset.

Taulukko 4.3: Kaikki testitapaukset.

Testitapaus	**Kuvaus**
AnswerNewestCall	Vastaa uusimpaan puheluun
AnswerOldestCall	Vastaa vanhimpaan puheluun
CloseIncidentForm	Sulkee ilmoituslomakkeen
SelectDispatchButton	Painaa ”Hälytä”-painiketta
SelectIncidentTypeAnimalRescue	Valitsee tehtävätyypiksi eläimen pelastus
SelectIncidentTypeBushFire	Valitsee tehtävätyypiksi maastopalo
SelectIncidentTypeFalling	Valitsee tehtävätyypiksi kaatuminen
SelectIncidentTypeGroundless	Merkitsee ilmoituksen aiheettomaksi
SelectRandomIncident	Valitsee satunnaisen vanhan tehtävän
SelectRoleIVO	Kirjautuu järjestelmään roolissa IVO
SelectUnitsEmergency	Valitsee ensihoidon yksiköitä
SelectUnitsPolice	Valitsee poliisiyksiköitä
SelectUnitsRescue	Valitsee pelastusyksiköitä
SelectUnitsSocialServices	Valitsee sosiaalihuollon yksiköitä
StartClient	Käynnistää sovelluksen

IVO on tehtävärooli, joka on lyhenne termistä ilmoitusten vastaanotto. IVO suorittaa ilmoitusten vastaanottamisen lisäksi tehtävään liittyvää riskinarviota, paikannuksen sekä tehtävän hälyttämisen ja sen välittämisen viranomaiselle.

4.4.1 Testit

Perinteisissä automaattisissa testeissä hyödynnettiin pientä osaa testitapauksista, jotta saatiin kohdistettua tutkimusta vain pienelle osalle järjestelmää. Taulukossa 4.4 testit ovat suoritusjärjestyksessä, jota käytettiin testauksissa. Testijoukot voivat sisältää muita testijoukkoja tai testitapauksia.

Taulukko 4.4: Testit suoritusjärjestyksessä.

Testijoukko	**Testi**
IVO - Stresstest	IVO - Login
	IVO - Answer Prank Call
IVO - Login	StartClient
	SelectRoleIVO
IVO - Answer Prank Call	AnswerNewestCall
	SelectIncidentTypeGroundless
	CloseIncidentForm

”IVO - Stresstest” -testijoukko koostuu kahdesta testijoukosta: ”IVO - Login” ja ”IVO - Answer Prank Call”. ”IVO - Login” -testijoukko suoritettiin vain kerran koko testauksen aikana.

Liitteessä 2 on kuvattu kaikki käytetyt mallipohjaisessa testauksessa käytetyt mallipohjaiset testijoukot ja niihin liittyvät testit. Taulukossa 4.5 ”IVO - Modelled tests”-testijoukko on koko mallipohjaisen testauksen juurijoukko, jonka avulla testaus suoritettiin kokonaisuudessaan.

Taulukko 4.5: Normaalit testijoukot.

Testijoukko	**Testit**
IVO - Incident Form	IVO - Risk Evaluation
	IVO - Add Units Condition
	IVO - Dispatch Condition
	CloseIncidentForm
IVO - Login	StartClient
	SelectRoleIVO
IVO - Main Screen	IVO - Main Operation
	IVO - Incident Form
IVO - Modelled tests	IVO - Login
	IVO - Main Screen

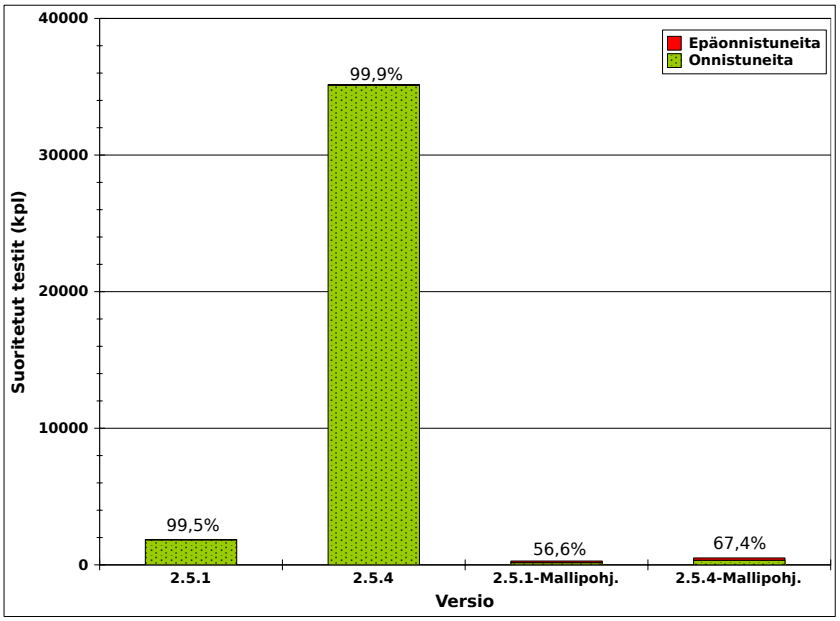
Mallipohjaisten testien määrittelyssä hyödynnettiin operationaalisia profiileja testijoukkojen muodostukseen. IVO-roolin ja hätäkeskuspäivystäjän sovelluksen eri käyttötilojen perusteella kehitettiin erilaisia operationaalisia profiileja. Todennäköisyyksiin perustuvia testijoukkoja saatiin muodostettua kahdeksan kappaletta ja ehtoihin perustuvia testijoukkoja viisi kappaletta.

4.4.2 Tulokset

Version 2.5.1 normaalien testien aikana havaittiin muistivuoto, mikä aiheutti testien hidastumisen ja lopulta sovellus kaatui. Versioiden 2.5.2 ja 2.5.3 aikana muistivuodon aiheuttajaa pyrittiin etsimään poistamalla pieniä kokonaisuuksia sovelluksesta ja haravoimaan, mikä aiheuttaisi muistivuodon. Lopulta muistivuodon aiheuttaja löytyi eräästä ilmoituslomakkeen komponentista. Kyseinen muistivuoto ja muita korjauksia tehtiin versioon 2.5.4.

Perinteisten testien versioiden 2.5.1 ja 2.5.4 välillä vasteajat parantuivat selkeästi. Esimerkiksi versiossa 2.5.1 puheluun vastaamisessa kesti keskiarvoltaan 4905 millisekuntia ja mediaaniltaan 4388 millisekuntia. Vastaavasti versiossa 2.5.4 puheluun vastaaminen oli keskiarvoltaan 3137 millisekuntia ja mediaaniltaan 2658 millisekuntia.

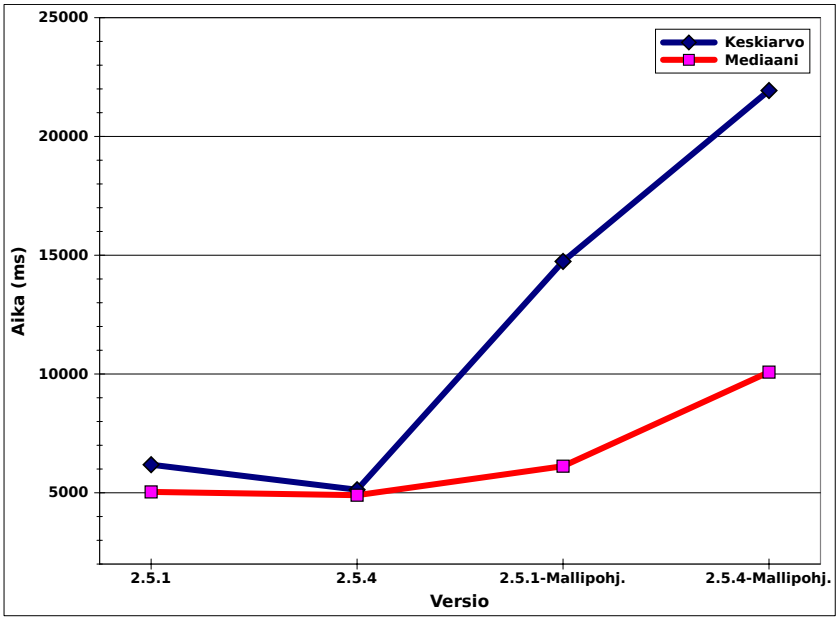
Kuvassa 4.7 on esitetty suoritettujen juuritestijoukkojen suoritusmäärät. Kuvassa on sekä perinteisten testien että mallipohjaisten testien suoritusmäärät. Jokaisen pylvään päällä olevat prosentit kertovat onnistuneiden testien osuutta kaikista suorituksista. Kuvasta nähdään, että mallipohjaiset testejä suoritettiin huomattavasti vähemmän ja onnistuneita testejä oli vähemmän kuin perinteisissä testeissä. Järjestelmän hitaus aiheutti testien epäonnistumisen, koska testeissä oli tarkastuspisteitä viiden sekunnin aikarajalla.



Kuva 4.7: Hätäkeskustietojärjestelmän testien juuritestijoukkojen suoritusmäärät.

Tästä voidaan todeta, että mallipohjaiset testit aiheuttavat huomattavasti enemmän kuormaa kuin perinteiset testit. Mallipohjaisilla testeillä pyritään saamaan aikaan vaihtelevuutta testisuorituksiin, jotta nähdään, miten järjestelmä kykenee toimimaan yllättävissäkin tilanteissa.

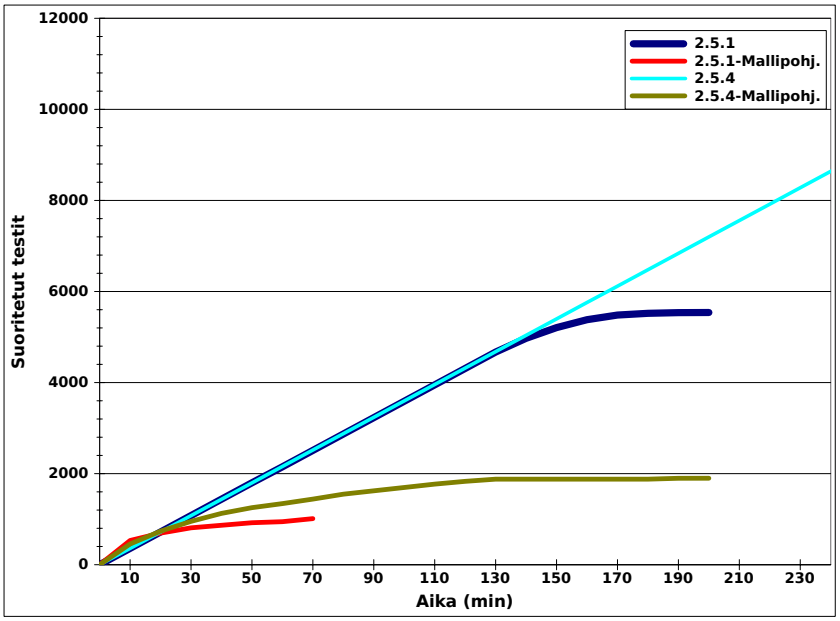
Kuvassa 4.8 on esitetty juuritestijoukkojen vasteajat molemmista versioista. Kuvassa on sekä perinteisten testien että mallipohjaisten testien vasteajat. Vasteajoista on otettu huomioon vain keskimääräinen ja mediaani. Kuten kuvasta huomaa, version 2.5.1 mallipohjaisten testien keskimääräinen suoritusaika oli huomattavasti korkeampi kuin perinteisissä testeissä. Odotetusti mallipohjaiset testit aiheuttivat paljon vaihtelevuutta testien suoritusjärjestykseen. Tämän takia mallipohjaiset testit hidastuivat ja aiheuttivat suurempaa kuormaa ennakoimattomalla tavalla.



Kuva 4.8: Hätäkeskustietojärjestelmän testien juuritestijoukkojen vasteajat.

Kuvassa 4.9 kuvataan kaikkien ajettujen testien kokonaissuoritukset suhteessa testipedin ajoaikaan. Kyseisestä kuvasta on rajattu pois version 2.5.4 tulosviivan loppuosa, koska se jatkuisi tasaisesti kasvavana aina 940 minuuttiin asti.

Kuvasta 4.9 huomaa, että versioiden 2.5.1 ja 2.5.4 välillä oli selkeä parannus perinteisten testien osalta. Kuten kuvasta näkee, versiossa 2.5.1 ollut muistivuoto hidasti testien suoritusta ja lopuksi kaatoi sovelluksen testauksen. Ensimmäisen kymmenen minuutin aikana version 2.5.1 mallipohjaiset testit olivat nopeampia kuin muut testit, kuten kuvasta nähdään. Kymmenen minuutin jälkeen järjestelmän tehokkuus heikkeni radikaalisti.



Kuva 4.9: Hätäkeskustietojärjestelmän testien suoritusmäärät suhteessa aikaan.

Perinteisten testien osalta on laskettu hätäkeskustietojärjestelmän saatavuus kaavan 3.3 mukaisesti, koska niiden onnistumisprosentit olivat lähinnä 100 prosenttia. Version 2.5.1 saatavuus oli 96,3 % ja vastaavasti version 2.5.4 99,8 %. Edellisten arvioiden mukaan uusi hätäkeskustietojärjestelmä ei vastaa vielä luvattua korkean saatavuuden tasoa, joka on 99,996 %. Lasketut saatavuudet ovat arvioita, joiden perusteella voidaan todeta, että testattava järjestelmä saattaa päästä luvattuun saatavuuteen.

4.4.3 Yhteenveto

Hätäkeskustietojärjestelmän perinteinen automaattinen testaus oli suoraviivainen prosessi, kuten Varotietopalvelun testauksessa. Prosessi eteni testien suunnittelusta suoritukseen ja testitulosten tulkinnasta arviointiin.

Mallipohjaisten testien suunnittelu osoittautui haastavaksi, koska operationaalisten profiilien suunnittelussa täytyi ottaa tarkkaan huomioon, etteivät ehdot olisi ristiriidassa keskenään. Kun operationaaliset profiilit saatiin muodostettua, testijoukkoja kehitettiin profiilien perusteella. Mallipohjaisten testijoukkojen kokoamisen aikana todettiin, että testien graafityökalu voisi edesauttaa monimutkaisempien testien muodostamista ja hahmottamista. Graafiominaisuudesta on jatkokehitysajatuksissa kohdassa 4.5.

Perinteiset testit aiheuttivat kevyempää kuormaa järjestelmälle, koska ne jäivät odottamaan aina uuden ilmoituksen syntyä. Tällöin testipeti jäi odottamaan ilmoitusta eikä aiheuttanut kuormaa muulla tavoin. Perinteisissä testeissä oli otettu huomioon pieni kokonaisuus järjestelmästä, mikä oli osittainen syy pieneen kuormaan.

Muistivuodon etsiminen vaati jatkuvasti muutoksia testattavaan sovellukseen, jotta voitiin löytää tehokkaasti muistivuotoa aiheuttanut kohta. Testien suorittaminen testipedillä uudelleen ei vaatinut ylimääräistä työtä, jolloin testit voitiin toistaa luotettavasti samalla tavalla ja tulokset olivat vertailukelpoisia keskenään.

Mallipohjaisilta testeiltä odotettiin, että ne vaikuttaisivat testien suoritukseen selvästi. Tässä onnistuttiin, koska mallipohjaiset testit eivät jääneet odottamaan järjestelmältä tiettyä tapahtumaa, kuten esimerkiksi ilmoitusjonossa tapahtuvaa muutosta. Perinteisessä automaattisissa testeissä testipeti jäi odottamaan simulaattorin luomaa ilmoitusta. Mallipohjaisilla testeillä pystyttiin luomaan käyttäjää simuloivia malleja, joiden avulla voidaan tutkia myöhemmin esimerkiksi harvinaisempia tilanteita, jotka olisivat työläämpiä suorittaa manuaalisella testauksella.

Hätäkeskustietojärjestelmän testaus on onnistunut molempien testitapojen perusteella. Mallipohjaiset testit aiheuttivat paljon vaihtelevuutta testien suoritusjärjestykseen, jolloin testien suoritus hidastui ja testit aiheuttivat suurempaa kuormaa ennakoimattomalla tavalla. Version 2.5.1 mallipohjaisissa testeissä muistivuotoa ei olisi ilmaantunut kuin vasta erittäin pitkän ajan jälkeen, koska järjestelmän pullon-

kaulaksi tulikin jokin muu komponentti kuin hätäkeskuspäivystäjän sovellus.

Perinteisellä automaattisella testauksella voidaan kohdistaa testausta tiettyyn järjestelmän osaan. Testaamalla pientä osaa järjestelmästä voidaan etsiä muistivuotoja tietyistä komponenteista tai hyödyntää kuormitus- ja rasitustestauksessa löytääksemme järjestelmän komponenteille raja-arvoja.

Uusi hätäkeskustietojärjestelmä ei vastaa vielä luvattua korkean saatavuuden tasoa, joka on 99,996 %. Testeissä saatavuus mitattiin korkeimmillaan 99,8 %:iin. Kehitettävä järjestelmä on keskeneräinen eikä sitä ole vielä optimoitu, mutta myöhemmin testipetiä voidaan hyödyntää viimeistellyn järjestelmän testaamiseen ja todentaa järjestelmälle annettujen lupausten täyttyminen.

4.5 Jatkokehitys

Tässä luvussa käsitellään karkealla tasolla korkean saatavuuden testipedin mahdollisia jatkokehitysajatuksia ja jatkotoimia. Osa jatkokehitysajatuksista on kehityksen aikana tulleita ideoita, joita voidaan mahdollisesti myöhemmin käyttää testipedin kehityksessä. Loppuosa ajatuksista aiotaan hyödyntää myöhemmin korkean saatavuuden mittaamiseksi, mutta niitä ei ehditty käyttämään tämän työn tutkimuksessa.

4.5.1 Testipedin jatkokehitys

Tässä kohdassa käsitellään karkealla tasolla testipetiin liittyviä jatkokehitysajatuksia. Ne ovat ominaisuusehdotuksia, joita saatetaan kehittää myöhemmin testipetiin.

Ajoitetut testit

Voidaan kuvitella seuraava skenaario: Länsi-Suomessa tulee raju myrsky, joka kaataa puita ja katkaisee sähköt seudun asukkailta. Tämän seurauksena hätäkeskukseen soittaa sadoittain ihmisiä ja aiheuttaa järjestelmälle rajun piikin.

Erilaiset epätavalliset tilanteet voivat aiheuttaa järjestelmässä hetkittäisesti suurta kuormaa, jolloin olisi hyvä tutkia, miten järjestelmä selviää suurista kuormamääristä hetkittäin. Tutkimista voisi avustaa ajastetut testit ja ajoitetut simulaattoritoiminnan säätelyt. Jos kehitettävää järjestelmää varten on tehty simulaattoreita, niin niitä voidaan käyttää epätavallisten tilanteiden luomiseen. Aiheuttamalla järjestelmään hetkittäisiä kuormapiikkejä voidaan nähdä, miten ne vaikuttavat esimerkiksi asiakassovelluksen tai järjestelmän eri palveluiden toimintaan.

Testipetien välinen yhteistyö

Kahden tai useamman eri käyttäjän tekemät muutokset voivat vaikuttaa muihin, kun järjestelmään on kytkeytynyt monia erilaisia käyttäjiä. Varsinkin reaaliaikai-

seen tietoon tai synkronointiin perustuvissa järjestelmissä, kuten ryhmätyösovelluksissa, voi tapahtua paljon konflikteja käyttäjien aiheuttamien muutoksien takia. Konfliktitilanteiden ratkaisu voidaan jättää käyttäjälle tai järjestelmälle itselleen.

Yhtenä jatkokehitysajatuksena on testipetien välisen yhteistyön rakentaminen. Tällöin voidaan tutkia, miten järjestelmä käyttäytyy ja kuinka luotettavasti, kun useampi testipeti tekee paljon muutoksia samalle asialle. Tätä voitaisiin kehittää edelleen sille asteelle, että yksi testipeti suorittaa testin ja välittää testauskäskyn muille ”alaisilleen”.

Esimerkiksi yksi testipeti suorittaa testin, joka luo aluksi uuden muistiinpanon ja pyytää toista testipetiä avaamaan sen omassa testissään. Tämän jälkeen ensimmäinen testipeti lisäisi tekstiä ja käskyttää toista muokkaamaan kyseistä tekstiä.

Kriittisten palveluiden tilojen mittaaminen

Koska tässä työssä enemmän keskityttiin korkean saatavuuden mittaamiseen käyttäjänäkökulmasta, olisi hyvä kuitenkin tutkia järjestelmän sisäiseen rakenteeseen ja eri palveluiden saatavuuteen. Esimerkiksi Miniclient voidaan toteuttaa palvelua käyttävänä simulaattorina, jolloin testit voidaan kohdistaa suoraan yksittäiseen palveluun.

Skenaariotyökaluun graafiominaisuus

Testipedin nopean kehitystahdin aikana ei panostettu käyttöliittymäominaisuuksiin, vaan haluttiin päästä todentamaan, että testipeti toimii odotusten mukaan. Kuitenkin useat mallipohjaiseen testaukseen pohjautuvat työkalut omaavat graafiominaisuuden, jonka pitäisi vähentää testaajan taakkaa testimallien ylläpidossa.

Graafien avulla voitaisiin määritellä erilaisia testauspolkuja, testeihin vaikuttavia muuttujia, tai mahdollisesti todennäköisyyksiä eri testeille.

4.5.2 Häiriöiden vaikutus korkeaan saatavuuteen

Tässä kohdassa esitetään erilaisia testaustapoja, joiden avulla voitaisiin arvioida järjestelmän korkeaa saatavuutta. Testaustavat keskittyvät enemmän harvinaisten tilanteiden luomiseen ja etenkin häiriön aiheuttamiseen testattavassa järjestelmässä.

Useiden testipedien ajaminen

Kun halutaan selvittää, miten testattava järjestelmä kykenee selviytymään kuorman aiheuttamasta häiriöstä, niin silloin sitä pitää tutkia useiden testipetien kuorman tuottamisella. Esimerkiksi useat testipedit suorittaisivat yhtä aikaa useampia testejä, jotka muuttavat paljon tietoa tietokannoissa.

Myöhemmin ajatusta voidaan jatkojalostaa esimerkiksi haavoittuvuuden testaamiseen. Tällöin voisi olla useampia testipetejä, jotka voisivat suorittaa sellaisia testejä, jotka voisivat aiheuttaa tai löytää tietoturva-aukkoja esimerkiksi injektioiden kautta tai tehdä palvelunestohyökkäyksiä.

Kriittisen järjestelmän häirintä

Eräs olennaisin osa haavoittuvuustestausta on kriittisen järjestelmän vikasietoisuuden testaus. Vikasietoisuutta voitaisiin testata esimerkiksi aiheuttamalla palveluihin häirintää verkkoyhteyksien hetkellisillä katkoksilla.

Muita häirintätapoja voisi olla muiden palveluiden tai palvelinprosessien sammuttaminen hetkellisesti, levytilan vähentäminen, ja kuorman aiheutus muissa prosesseissa. Satunnaisella häiriön aiheuttamisella voidaan aiheuttaa järjestelmään katkoksia ja tutkia, miten se selviää erilaisista häiriötilanteista.

4.5.3 Kuormitustestaus

Kuten kohdassa 2.3.2 todettiin, kuormitustestauksen ideana on suorittaa testejä tasaisella, mutta realistisella kuormalla. Tämä kuitenkin voi vaatia testattavalta järjestelmältä vakautta ennen kuin kuormitustestausta voidaan suorittaa.

Kuormitustestausta voidaan hiljalleen kasvattaa ja samalla nostaa tavoitteita, kun edelliset haasteet on jo saavutettu. Esimerkiksi voitaisiin pitää 10 % kuormaa realistisesta määrästä lähtötilanteena. Jos testit läpäistään kirkkaasti läpi tietyssä ajassa, voidaan nostaa kuorman tasoa 10 %:lla.

Kuitenkaan kuormitustestausta ei kannata lopettaa 100 %:n kuormitustestien jälkeen, vaan sitä tulisi jatkaa niin pitkälle kuin on mahdollista. Realistisen kuorman ylityksellä voidaan varmentaa järjestelmän toimivuutta myös hetkittäisten kuormapiikkien suhteen, tai pitkällä aikavälillä järjestelmän kykyä skaalautua kuorman suhteen.

4.5.4 Rasitustestaus

Kohdassa 2.3.3 määritettiin, että rasitustestaus on testaustapa, jossa pyritään hakemaan raja-arvoja kuormittamalla järjestelmää mahdollisimman suurella kuormalla. Kuormittamalla järjestelmän yhtä palvelua tai komponenttia saadaan kyseisille osille tarkat raja-arvot, millä rasituksella testatun osan tulisi kestää. Hätäkeskustietojärjestelmän testituloksissa todettiin, että perinteisillä automaattisilla testeillä voidaan mitata yhden osan raja-arvoja.

5. JOHTOPÄÄTÖKSET

Tämän työn tutkimusongelmana oli, miten järjestelmien korkeaa saatavuutta voidaan mitata. Tietojärjestelmien suunnittelun yhteydessä saatavuus määritellään palvelutasosopimuksissa, jotka määrittävät kuinka saatavissa palvelun pitäisi olla.

Tutkimusongelmaa pyrittiin ratkaisemaan automaattisella testauksella, jonka avulla mitattiin testattavan järjestelmän saatavuutta. Mittaus tehtiin korkean saatavuuteen liittyvien laatutekijöiden näkökulmasta. Saatavuuden lisäksi mitattavat laatutekijät olivat luotettavuus, tehokkuus, skaalautuvuus ja ylläpidettävyys.

Tässä työssä hyödynnettiin toiminnallista testausta, haavoittuvuus-, suorituskyky-, harmaalaatikkotestausta ja mallipohjaista testausta. Työssä keskityttiin kuorman tuottamiseen oikeellisten testien avulla ja haavoittuvuustestausta tehtiin testien satunnaistamisella. Mallipohjaisilla testeillä pyrittiin jäljittelemään hätäkeskuspäivystäjän toimia, jolloin voidaan tutkia korkeaa saatavuutta lähes oikeiden skenaarioiden valossa.

Työssä onnistuttiin arvioimaan käyttäjäkokemukseen perustuvaa korkeaa saatavuutta, jossa mittaus tehtiin toiminnon aloituksesta järjestelmän antamaan vasteeseen asti. Mittaus perustui siihen, ettei käyttäjän tarvitse olla tietoinen siitä, mitä järjestelmässä tapahtuu, jos sen kriittiset palvelut ovat kykeneviä toimimaan niin kuin pitäisi. Käyttäjät ovat kuitenkin kehitettävien järjestelmien käytön kannalta tärkein sidosryhmä.

Tämän työn aikana kehitettiin testipeti, jonka avulla pyrittiin mittaamaan edellä mainittuja laatutekijöitä objektiivisesti. Testipedin kehityksen aikana kohdattiin monia haasteita resursoinnin ja priorisoinnin osalta. Eniten aikaa kehityksen lisäksi kului tulosten analysointiin ja raportointiin. Automaattisen testauksen kehittäminen vaatii paljon aikaa ja resursseja, mikä huomattiin testipedin kehityksessä.

Kehityksen aikana voitiin keksiä uusia ominaisuuksia testipedille, joilla parannettiin muun muassa testitulosten luettavuutta. Ketterän kehityksen ansiosta pystyttiin testipetiä hyödyntämään jo varhaisessa vaiheessa. Kaikkia ominaisuuksia ei voida kehittää tällaiselle työkalulle, koska se voi monimutkaistaa testipedin rakennetta ja sen käyttö tulisi haastavammaksi.

Testipedin kehittämisestä oli paljon hyötyä, koska saatiin sellainen työkalu, jonka avulla voidaan testata järjestelmän korkeaa saatavuutta eri laatutekijöiden näkökulmista. Testipeti täytti kehityksen alkuvaiheessa muodostuneet odotukset ja si-

tä kehitetään edelleen. Testipeti on onnistunut myös tuotteen näkökulmasta, koska sillä voidaan testata Java-ohjelmointikielellä kehitettyjä sovelluksia eikä se rajoita testausta millään tavalla. Testien määrittelijältä vaaditaan laajaa tietämystä testattavasta järjestelmästä saadakseen testipedistä täyden hyödyn.

Työssä onnistuttiin siis kehittämään saatavuuden mittaukseen erikoistunut testipeti, jonka avulla löydettiin testattavasta järjestelmästä virheitä. Kyseisiä virheitä ei olisi löydetty manuaalisella testauksella yhtä tehokkaasti. Jos testipedillä oli suoritettu aamupäivän ajan testejä, testeissä löytyneitä virheitä saatettiin korjata jo samana iltapäivänä.

Tässä työssä testauskohteena olivat uusi kehitettävä hätäkeskustietojärjestelmä ERICA ja siihen liitettävä Varotietopalvelu. Varotietopalvelua testattiin kuusi kertaa ja hätäkeskustietojärjestelmää neljä kertaa. Mallipohjaista testausta hyödynnettiin hätäkeskustietojärjestelmän testauksessa kahdesti.

Testien suunnittelu ja kehittäminen osoittautui suoraviivaiseksi perinteisen automaattisen testauksen osalta. Haastavin osuus oli mallipohjaisten testien suunnittelu, koska siinä käytettiin operationaalisia profiileja. Operationaalisten profiilien suunnittelussa täytyi huomioida, etteivät ehdot olisi ristiriidassa keskenään.

Varotietopalvelun testauksessa löydettiin järjestelmästä virheitä ja tehottomia toimintoja. Virheelliset toiminnot korjattiin ja tietokantaan liittyvät optimoinnit tehtiin tulosten perusteella. Varotietopalvelun luotettavuutta ja tehokkuutta saatiin parannettua huomattavasti testitulosten perusteella.

Hätäkeskustietojärjestelmän testeissä havaittiin, että perinteinen automaattinen testaus ja mallipohjainen testaus ovat tehokkaita korkean saatavuuden mittaamisessa. Perinteiset automaattiset testit aiheuttivat kevyttä kuormaa järjestelmälle, mutta niiden avulla pystyttiin kohdistamaan testausta vain pienelle osalle järjestelmää. Perinteisillä automaattisilla testeillä päästiin sovelluksessa olevan muistivuodon jäljille. Testaamalla pientä osaa järjestelmästä voidaan etsiä muistivuotoja tietyistä komponenteista tai hyödyntää sitä kuormitus- ja rasitustestauksessa löytääksemme raja-arvoja järjestelmästä.

Mallipohjaisten testien avulla pystyttiin luomaan käyttäjää muistuttavia testimalleja, jotka kuvasivat todellisia käyttäjätoimintoja. Testien suoritusjärjestyksen vaihtelevuus aiheutti suurta kuormaa testattavaan järjestelmään.

Uusi hätäkeskustietojärjestelmä ei vastaa vielä luvattua korkean saatavuuden tasoa, joka on 99,996 %. Testeissä saatavuus mitattiin korkeimmillaan 99,8 %:iin. Kehitettävä järjestelmä on keskeneräinen eikä sitä ole vielä optimoitu, mutta myöhemmin testipetiä voidaan hyödyntää viimeistellyn järjestelmän testaamiseen ja todentaa järjestelmälle annettujen lupausten täyttymisen.

Testipedillä pystyttiin suorittamaan kestotestausta luotettavasti, mutta testausautomaatio ei ole korkean saatavuuden testauksessakaan hopea luoti, koska testi-

tulokset voivat olla osittain virheellisiä. Tämä havaittiin Varotietopalvelun testauksessa, jossa testipeti oli jumiutunut kesken testien. Testipedin luotettavuutta voidaan mitata testaamalla esimerkiksi testipedin rakennetta kestotestauksen muodossa. Tällä voidaan katsoa, että testipeti kykenee tyhjilläkin testeillä suoriutumaan useamman päivän ajan, sekä voidaan varmistaa, ettei testipeti sisällä virheitä.

Tässä työssä pääpaino oli testipedin kehityksessä sekä siinä, että voidaanko sen avulla mitata korkeaa saatavuutta. Mittaus perustui käyttäjän kokemaan saatavuuteen ja testaukseen valikoitiin vain oleellisimmat järjestelmän toiminnot, joita vasten suoritettiin pitkäkestoisia testejä. Testituloksia on arvioitava huolellisesti, koska testaukset suoritettiin tietyllä aikavälillä ja mittaus on arvio, pääseekö järjestelmä palvelutasosopimuksissa asetettuihin lupauksiin. Todellisuudessa korkean saatavuuden järjestelmien täytyy olla toimintakykyisiä kuukausien ajan ja testauksella voidaan vain saada arvioita järjestelmän toimintakyvystä. Saatuja arvioita voisi tarkentaa kuormitus- ja rasitustestauksella, joihin ei keskitytty tässä työssä.

Työn aikana havaittiin, että yksi korkean saatavuuden mittaamiseen liittyvä ongelma on testausajan pituus. Liian lyhyellä aikavälillä ei välttämättä löydetä kunnon ongelmia, ja se voisi antaa liian hyvän kuvan mitattavasta järjestelmästä. Esimerkiksi perinteisten automaattisten testien aikana löydetty muistivuoto olisi vaatinut mallipohjaisten testien suoritusta huomattavasti pidempään, koska järjestelmän ylikuormittuminen olisi peittänyt kyseisen muistivuodon. Testausajan pidennys voi vaikuttaa muihin aikarajoihin negatiivisesti, jolloin esimerkiksi ongelmia ei ehditä korjaamaan ennen toimitusajankohtaa. Järjestelmän tietokantojen vanhentamisella voidaan lyhentää testausaikaa, jos voidaan olettaa järjestelmän muiden osien toimivan oikein ajasta riippumatta.

Korkean saatavuuden mittaaminen ei ole yksinkertaista verrattuna yksittäisten laatuominaisuuksien mittaamiseen, sillä on otettava huomioon kaikki korkeaan saatavuuteen liittyvät laatuominaisuudet ja mitattava järjestelmä kokonaisuutena. Tällöin ei välttämättä löydetä yksittäisiä piileviä ongelmia järjestelmän eri osissa, vaan ne saattaa löytyä vasta ylläpitovaiheessa. Jos ongelmia ilmenee ylläpitovaiheessa, niin niiden korjaus on huomattavasti kalliimpaa kuin kehitysvaiheessa löydettyjen ongelmien korjaaminen. Suureen korjaushintaan vaikuttaa se, kuinka kauan järjestelmä on pois käytöstä.

Korkeaa saatavuutta täytyy mitata mahdollisimman varhaisessa vaiheessa, jotta toimitettavasta järjestelmästä on korjattu suurin osa vakavista virheistä ja järjestelmän laatu on varmistettu. Samalla voidaan varmentua siitä, että järjestelmä läpäisee kaikki palvelutasosopimuksen asettamat vaatimukset ja järjestelmä voidaan toimittaa asiakkaalle.

LÄHTEET

[1] Sutton, M., Greene, A. & Amini, P. Fuzzing: brute force vulnerabilty discovery. Safari Books Online, Addison-Wesley, 2007, 543 p.

[2] Patton, R. Software Testing. Sams Publishing, 2001, 389 p.

[3] Molyneaux, I. The Art of Application Performance Testing: Help for Programmers and Quality Assurance. Theory in practice, O'Reilly Media, 2009, 160 p.

[4] Schmidt, K. High Availability and Disaster Recovery: Concepts, Design, Implementation. Springer, 2006, 417 p.

[5] Meier, J.D., Homer, A., Hill, D., Taylor, J., Bansode, P., Wall, L., Boucher, R. & Bogawat, A. Microsoft Application Architecture Guide 2.0. 2009 [Viitattu 9.3.2013]. Saatavissa: http://www.microsoft.com/en-us/download/-details.aspx?id=16236.

[6] Utting, M. & Legeard, B. Practical Model-Based Testing: A Tools Approach. Morgan Kaufmann Publishers Inc., 2007, 456 p.

[7] Broekman, B. & Notenboom, E. Testing embedded software. Pearson Education Limited, 2003, 348 p.

[8] Ladin, R., Liskov, B., Shrira, L. & Ghemawat, S. Providing high availability using lazy replication. ACM Transactions on Computer Systems 10(1992)(4), pp 360–391. Saatavissa: http://doi.acm.org/10.1145/138873.138877.

[9] Gnanasekaran, V. Evaluating Application Architecture, Quantitatively, 2010 [Viitattu 9.3.2013]. Saatavissa: http://msdn.microsoft.com/en-us/-architecture/ff476939.aspx.

[10] Piedad, F. & Hawkins, M.W. High Availability: Design, Techniques, and Processes. Enterprise Computing Series, Prentice Hall, 2001, 266 p.

[11] Tang, D. & Hecht, H. An approach to measuring and assessing dependability for critical software systems. Software Reliability Engineering, 1997. Proceedings., The Eighth International Symposium on. 1997, pp 192–202.

[12] Lewis, W.E. Software Testing and Continuous Quality Improvement. Auerbach Publications, kolmas painos, 2009. Saatavissa: http://common.-books24x7.com/toc.aspx?bookid=26474.

[13] Dustin, E., Rashka, J. & Paul, J. Automated Software Testing: Introduction, Management, and Performance. Addison-Wesley Professional, 1999, 608 p.

[14] Collins, E., Dias-Neto, A. & de Lucena, V.F. Strategies for Agile Software Testing Automation: An Industrial Experience. Computer Software and Applications Conference Workshops (COMPSACW), 2012 IEEE 36th Annual. 2012, pp 440–445.

[15] Craig, R. & Jaskiel, S. Systematic software testing. Artech House Publishers, 2002, 512 p.

[16] Melzer, I. Testing of a Computer Program on the Example of a Medical Application with Diversification and other Methods, 1996 [Viitattu: 24.2.2013]. Saatavissa: http://www.mathematik.uni-ulm.de/~melzer/thesis/thesis.html.

[17] Puolitaival, O.P. & Kanstrén, T. Mallipohjainen testaus ennen, nyt ja tulevaisuudessa. Teknologian tutkimuskeskus VTT, 8.11.2010 [Viitattu: 10.3.2013]. Saatavissa: http://www.vtt.fi/inf/julkaisut/muut/2010/mallipohjainentestaus.pdf.

[18] Musa, J.D. Software Reliability Engineering: More Reliable Software, Faster Development and Testing. McGraw-Hill, 1998, 391 p.

[19] Schieferdecker, I. Model-Based Fuzz Testing. Software Testing, Verification and Validation (ICST), 2012 IEEE Fifth International Conference on. 2012, pp 814.

[20] CWE. 2011 CWE/SANS Top 25 Most Dangerous Software Errors, 13.9.2011 [Viitattu: 24.2.2013]. Saatavissa: http://cwe.mitre.org/top25/index.html.

[21] Pelastustoimi. Uusi hätäkeskustietojärjestelmä ERICA otetaan käyttöön vuonna 2015, 19.11.2012 [Viitattu: 27.3.2013]. Saatavissa: http://www.pelastustoimi.fi/uutiset/6334.

LIITE 1: VAROTIETOPALVELUN TESTITULOKSET

Versio 2.2.3 testitulokset

Taulukko L1.1: Käytetyt tekniikat ja muutokset.

Tekniikat	
Tietokantajärjestelmä	H2
Yhteystekniikka	RMI

Taulukko L1.2: Testimäärät.

Suoritetut testit		
Testin nimi	**Onnistuneet**	**Epäonnistuneet**
CautionInformation	2806 (86 %)	443 (14 %)
CreateAddress	3249 (100 %)	0
CreatePerson	3249 (100 %)	0
CreateVehicle	3249 (100 %)	0
ModifyAddress	3086 (100 %)	0
ModifyPerson	3239 (100 %)	0
ModifyVehicle	3001 (100 %)	0
RemoveAddress	3125 (96 %)	124 (4 %)
RemovePerson	3248 (100 %)	0
RemoveVehicle	2810 (86 %)	438 (14 %)

Taulukko L1.3: Testien vasteajat.

Testien vasteajat				
Testin nimi	**Suurin**	**Pienin**	**Keskiarvo**	**Mediaani**
CautionInformation	100006	197	22030	9765
CreateAddress	32559	42	6875	2917
CreatePerson	23257	15	3752	1482
CreateVehicle	25220	34	5586	2652
ModifyAddress	1467	8	88	94
ModifyPerson	1389	8	109	94
ModifyVehicle	1404	7	77	63
RemoveAddress	11189	4	2356	843
RemovePerson	1922	8	117	93
RemoveVehicle	11263	1	3075	1529

Versio 2.2.4 testitulokset

Taulukko L1.4: Käytetyt tekniikat ja muutokset.

Tekniikat ja muutokset	
Tietokantajärjestelmä	H2
Yhteystekniikka	RMI
Muutokset	Tietokannan persistoinnin optimointi

Taulukko L1.5: Testimäärät.

Suoritetut testit		
Testin nimi	**Onnistuneet**	**Epäonnistuneet**
CautionInformation	2837 (90 %)	311 (10 %)
CreateAddress	3149 (100 %)	0
CreatePerson	3149 (100 %)	0
CreateVehicle	3149 (100 %)	0
ModifyAddress	3149 (100 %)	0
ModifyPerson	3149 (100 %)	0
ModifyVehicle	3149 (100 %)	0
RemoveAddress	3093 (98 %)	55 (2 %)
RemovePerson	3149 (100 %)	0
RemoveVehicle	2842 (90 %)	306 (10 %)

Taulukko L1.6: Testien vasteajat.

Testien vasteajat				
Testin nimi	**Suurin**	**Pienin**	**Keskiarvo**	**Mediaani**
CautionInformation	95444	188	19526	9095
CreateAddress	30790	41	6086	2719
CreatePerson	20270	18	3199	1378
CreateVehicle	24271	34	4989	2421
ModifyAddress	1417	7	83	93
ModifyPerson	1514	9	109	95
ModifyVehicle	1280	7	68	46
RemoveAddress	10807	1	2096	783
RemovePerson	1526	9	109	88
RemoveVehicle	11249	5	2804	1428

Versio 2.2.5 testitulokset

Taulukko L1.7: Käytetyt tekniikat ja muutokset.

Tekniikat ja muutokset	
Tietokantajärjestelmä	H2
Yhteystekniikka	RMI
Muutokset	Tallennus tietokantaan korjattu Osoitevarotiedon poistotoiminto korjattu Ajoneuvovarotiedon poistotoiminto korjattu

Taulukko L1.8: Testimäärät.

Suoritetut testit		
Testin nimi	**Onnistuneet**	**Epäonnistuneet**
CautionInformation	40003 (99,9 %)	28 (0,1 %)
CreateAddress	40023 (99,9 %)	9 (0,1 %)
CreatePerson	40017 (99,9 %)	15 (0,1 %)
CreateVehicle	40020 (99,9 %)	12 (0,1 %)
ModifyAddress	40032 (100 %)	0
ModifyPerson	40021 (99,9 %)	11 (0,1 %)
ModifyVehicle	40029 (99,99 %)	2 (0,01 %)
RemoveAddress	40019 (99,9 %)	12 (0,1 %)
RemovePerson	40022 (99,9 %)	9 (0,1 %)
RemoveVehicle	40013 (99,9 %)	15 (0,1 %)

Taulukko L1.9: Testien vasteajat.

Testien vasteajat				
Testin nimi	**Suurin**	**Pienin**	**Keskiarvo**	**Mediaani**
CautionInformation	95011	145	4300	2242
CreateAddress	24394	33	935	447
CreatePerson	18519	15	428	217
CreateVehicle	29040	27	720	324
ModifyAddress	6303	5	21	9
ModifyPerson	73314	7	635	152
ModifyVehicle	6233	1	25	9
RemoveAddress	13156	1	346	158
RemovePerson	37177	9	900	282
RemoveVehicle	13571	1	288	156

Versio 2.2.7 testitulokset

Taulukko L1.10: Käytetyt tekniikat ja muutokset.

Käytetyt tekniikat ja muutokset	
Tietokantajärjestelmä	PostgreSQL
Yhteystekniikka	JMS
Muutokset	Tietokantajärjestelmä vaihdettu Yhteystekniikka vaihdettu

Taulukko L1.11: Testimäärät.

Suoritetut testit		
Testin nimi	**Onnistuneet**	**Epäonnistuneet**
CautionInformation	25355 (99 %)	47 (1 %)
CreateAddress	25396 (99,9 %)	7 (0,1 %)
CreatePerson	25399 (99,9 %)	5 (0,1 %)
CreateVehicle	25396 (99,9 %)	6 (0,1 %)
ModifyAddress	25400 (99,99 %)	2 (0,01 %)
ModifyPerson	25398 (99,9 %)	4 (0,1 %)
ModifyVehicle	25399 (99,99 %)	1 (0,01 %)
RemoveAddress	25388 (99,9 %)	13 (0,1 %)
RemovePerson	25400 (99,99 %)	2 (0,01 %)
RemoveVehicle	25393 (99,9 %)	7 (0,1 %)

Taulukko L1.12: Testien vasteajat.

Testien vasteajat				
Testin nimi	**Suurin**	**Pienin**	**Keskiarvo**	**Mediaani**
CautionInformation	60132604	387	6003	2195
CreateAddress	15650	89	768	421
CreatePerson	20503	45	472	224
CreateVehicle	20911	78	764	347
ModifyAddress	12997	16	107	31
ModifyPerson	13489	26	270	122
ModifyVehicle	14547	15	85	31
RemoveAddress	60132032	1	2749	177
RemovePerson	14802	24	419	229
RemoveVehicle	21329	1	363	176

Versio 2.2.20 testitulokset

Taulukko L1.13: Käytetyt tekniikat ja muutokset.

Käytetyt tekniikat ja muutokset	
Tietokantajärjestelmä	PostgreSQL
Yhteystekniikka	JMS
Muutokset	Lukuisia korjauksia tehty version 2.2.7 jälkeen

Taulukko L1.14: Testimäärät.

Suoritetut testit		
Testin nimi	**Onnistuneet**	**Epäonnistuneet**
CreateAllInformation	57481 (98 %)	853 (2 %)
ModifyAddress	57479 (98 %)	854 (2 %)
ModifyPerson	57480 (98 %)	853 (2 %)
ModifyVehicle	57479 (98 %)	854 (2 %)
NewCautionInformation	57477 (98 %)	856 (2 %)
RemoveAddress	57478 (98 %)	855 (2 %)
RemovePerson	57479 (98 %)	854 (2 %)
RemoveVehicle	57478 (98 %)	855 (2 %)

Taulukko L1.15: Testien vasteajat.

Testien vasteajat				
Testin nimi	**Suurin**	**Pienin**	**Keskiarvo**	**Mediaani**
CreateAllInformation	44324	48	1022	320
ModifyAddress	30112	5	471	12
ModifyPerson	30023	12	480	13
ModifyVehicle	30031	7	468	12
NewCautionInformation	212127	127	4048	476
RemoveAddress	30097	6	533	27
RemovePerson	30056	8	499	29
RemoveVehicle	32100	6	568	16

VTP-testaustulokset 2.2.21

Taulukko L1.16: Käytetyt tekniikat ja muutokset.

Käytetyt tekniikat ja muutokset	
Tietokantajärjestelmä	PostgreSQL
Yhteystekniikka	JMS
Muutokset	Tietokantaoperaatiot korjattu

Taulukko L1.17: Testimäärät.

Suoritetut testit		
Testin nimi	**Onnistuneet**	**Epäonnistuneet**
CreateAllInformation	57438 (99,9 %)	37 (0,1 %)
ModifyAddress	57440 (99,9 %)	35 (0,1 %)
ModifyPerson	57440 (99,9 %)	35 (0,1 %)
ModifyVehicle	57439 (99,9 %)	36 (0,1 %)
NewCautionInformation	57424 (99,9 %)	51 (0,1 %)
RemoveAddress	57438 (99,9 %)	37 (0,1 %)
RemovePerson	57439 (99,9 %)	36 (0,1 %)
RemoveVehicle	57430 (99,9 %)	45 (0,1 %)

Taulukko L1.18: Testien vasteajat.

Testien vasteajat				
Testin nimi	**Suurin**	**Pienin**	**Keskiarvo**	**Mediaani**
CreateAllInformation	40613	46	608	315
ModifyAddress	33124	5	53	12
ModifyPerson	40601	6	70	14
ModifyVehicle	30022	4	52	12
NewCautionInformation	210150	127	1142	484
RemoveAddress	30011	8	121	29
RemovePerson	30005	7	82	30
RemoveVehicle	30105	5	150	16

LIITE 2: MALLIPOHJAISET TESTIT

Taulukko L2.1: Todennäköisyyteen perustuvat testijoukot.

Testijoukko	**Todennäköisyys**	**Testit**
IVO - Add Units Probability	0.7	IVO - Choose Unit Type
IVO - Answer Or Open Old Incident 1	0.6	AnswerOldestCall
	0.4	SelectRandomIncident
IVO - Answer Or Open Old Incident 2	0.9	AnswerOldestCall
	0.1	SelectRandomIncident
IVO - Choose Unit Type	0.25	SelectUnitsPolice
	0.25	SelectUnitsEmergency
	0.25	SelectUnitsRescue
	0.25	SelectUnitsSocialServices
IVO - Dispatch Units Probability	0.6	SelectDispatchButton
IVO - New Incident	0.7	IVO - Select Type Of Incident
	0.3	SelectIncidentTypeGroundless
IVO - Old Incident Probability	0.4	IVO - Select Type Of Incident
	0.1	SelectIncidentTypeGroundless
IVO - Select Type Of Incident	0.34	SelectIncidentTypeBushFire
	0.33	SelectIncidentTypeFalling
	0.33	SelectIncidentTypeAnimalRescue

Taulukko L2.2: Ehtoihin perustuvat testijoukot.

Testijoukko	**Ehdot**	**Testit**
IVO - Add Units Condition	Ilmoitus ei ole aiheeton	IVO - Add Units Probability
IVO - Dispatch Units Condition	Valittuja yksiköitä > 0	IVO - Dispatch Units Probability
IVO - Main Operation	Vanhoja ilmoituksia $= 0$	AnswerOldestCall
	Ilmoituksia jonossa $= 0$ Vanhoja ilmoituksia > 0	SelectRandomIncident
	$0 <$ Ilmoituksia ≤ 5 Vanhoja ilmoituksia > 0	IVO - Answer Or Open Old Incident 1
	Ilmoituksia > 5 Vanhoja ilmoituksia > 0	IVO - Answer Or Open Old Incident 2
IVO - Old Incident Condition	Ilmoitus ei ole aiheeton	IVO - Old Incident Probability
IVO - Risk Evaluation	Riskinarviota ei ole tehty	IVO - New Incident
	Riskinarvio on tehty	IVO - Old Incident Condition

LIITE 3: HÄTÄKESKUSTIETOJÄRJESTELMÄN TESTITULOKSET

Versio 2.5.1 testitulokset

Taulukko L3.1: Testimäärät.

Suoritetut testit		
Testin nimi	**Onnistuneet**	**Epäonnistuneet**
AnswerNewestCall	1841 (99 %)	5 (1 %)
CloseIncidentForm	1839 (99 %)	6 (1 %)
IVO - Answer Prank Call	1835 (99 %)	10 (1 %)
IVO - Login	1 (100 %)	0
IVO - Stresstest	1835 (99 %)	10 (1 %)
SelectIncidentTypeGroundless	1846 (100 %)	0
SelectRoleIVO	1 (100 %)	0
StartClient	1 (100 %)	0

Taulukko L3.2: Testien vasteajat.

Testien vasteajat				
Testin nimi	**Suurin**	**Pienin**	**Keskiarvo**	**Mediaani**
AnswerNewestCall	92412	785	4905	4388
CloseIncidentForm	232605	115	715	267
IVO - Answer Prank Call	382823	1279	6069	4978
IVO - Login	4690	4690	4690	4690
IVO - Stresstest	385620	1328	6183	5037
SelectIncidentTypeGround.	80847	51	510	337
SelectRoleIVO	2979	2979	2979	2979
StartClient	1697	1697	1697	1697

Versio 2.5.1 mallipohjaisten testien tulokset

Taulukko L3.3: Testimäärät.

Suoritetut testit		
Testin nimi	**Onnistuneet**	**Epäonnistuneet**
AnswerOldestCall	92 (59 %)	63 (41 %)
CloseIncidentForm	241 (86 %)	40 (14 %)
IVO - Add Units Condition	268 (100 %)	0
IVO - Add Units Probability	204 (100 %)	0
IVO - Answer Or Open Old Incident	140 (59 %)	99 (41 %)
IVO - Choose Unit Type	166 (100 %)	0
IVO - Dispatch Units Condition	281 (100 %)	0
IVO - Dispatch Units Probability	110 (100 %)	0
IVO - Incident Form	215 (77 %)	66 (23 %)
IVO - Login	1 (100 %)	0
IVO - Main Operation	183 (65 %)	100 (35 %)
IVO - Main Screen	159 (57 %)	122 (43 %)
IVO - Modelled tests	159 (57 %)	122 (43 %)
IVO - New Incident	143 (97 %)	5 (3 %)
IVO - Old Incident Condition	83 (77 %)	25 (23 %)
IVO - Old Incident Probability	49 (66 %)	25 (34 %)
IVO - Risk Evaluation	251 (89 %)	30 (11 %)
IVO - Select Type Of Incident	121 (80 %)	30 (20 %)
SelectDispatchButton	82 (100 %)	0
SelectIncidentTypeAnimalRescue	54 (87 %)	8 (13 %)
SelectIncidentTypeBushFire	34 (77 %)	10 (23 %)
SelectIncidentTypeFalling	33 (73 %)	12 (27 %)
SelectIncidentTypeGroundless	45 (100 %)	0
SelectRandomIncident	91 (71 %)	37 (29 %)
SelectRoleIVO	1 (100 %)	0
SelectUnitsEmergency	39 (100 %)	0
SelectUnitsPolice	47 (100 %)	0
SelectUnitsRescue	35 (100 %)	0
SelectUnitsSocialServices	45 (100 %)	0
StartClient	1 (100 %)	0

Taulukko L3.4: Testien vasteajat.

Testien vasteajat				
Testin nimi	**Suurin**	**Pienin**	**Keskiarvo**	**Mediaani**
AnswerOldestCall	8148	154	3133	3199
CloseIncidentForm	7971	101	1011	169
IVO - Add Units Condition	112537	1	1678	343
IVO - Add Units Probability	112525	1	2182	581
IVO - Answer Or Open Old Incident	395483	129	6946	4004
IVO - Choose Unit Type	112488	156	2654	715
IVO - Dispatch Units Condition	5795	51	755	124
IVO - Dispatch Units Probability	359	1	63	57
IVO - Incident Form	117821	220	8577	2572
IVO - Login	4705	4705	4705	4705
IVO - Main Operation	395555	199	6076	2200
IVO - Main Screen	411431	572	14700	6101
IVO - Modelled tests	411465	587	14739	6120
IVO - New Incident	39249	51	6021	1541
IVO - Old Incident Condition	87341	1	5092	24
IVO - Old Incident Probability	87323	1	7327	744
IVO - Risk Evaluation	87354	11	5150	776
IVO - Select Type Of Incident	87293	52	8842	2212

Testien vasteajat				
Testin nimi	**Suurin**	**Pienin**	**Keskiarvo**	**Mediaani**
SelectDispatchButton	358	51	82	69
SelectIncidentTypeAnimal-Rescue	55442	143	8265	1526
SelectIncidentTypeBushFire	87293	51	7446	1400
SelectIncidentTypeFalling	39075	816	11001	5489
SelectIncidentTypeGround-less	5293	51	2101	109
SelectRandomIncident	395448	103	9329	723
SelectRoleIVO	2959	2959	2959	2959
SelectUnitsEmergency	6714	183	1882	754
SelectUnitsPolice	112488	156	5001	611
SelectUnitsRescue	5403	192	1660	580
SelectUnitsSocialServices	5405	227	1642	764
StartClient	1730	1730	1730	1730

Versio 2.5.4 testitulokset

Taulukko L3.5: Testimäärät.

Suoritetut testit		
Testin nimi	**Onnistuneet**	**Epäonnistuneet**
AnswerNewestCall	35113 (99 %)	42 (1 %)
CloseIncidentForm	35155 (100 %)	0 (0 %)
IVO - Answer Prank Call	35113 (99 %)	42 (1 %)
IVO - Login	1 (100 %)	0 (0 %)
IVO - Stresstest	35113 (99 %)	42 (1 %)
SelectIncidentTypeGroundless	35155 (100 %)	0 (0 %)
SelectRoleIVO	1 (100 %)	0 (0 %)
StartClient	1 (100 %)	0 (0 %)

Taulukko L3.6: Testien vasteajat.

Testien vasteajat				
Testin nimi	**Suurin**	**Pienin**	**Keskiarvo**	**Mediaani**
AnswerNewestCall	15575	550	3137	2658
CloseIncidentForm	16157	101	351	323
IVO - Answer Prank Call	40644	726	3745	3417
IVO - Login	5433	5433	5433	5433
IVO - Stresstest	43977	1094	5128	4899
SelectIncidentTypeGround.	14592	51	254	206
SelectRoleIVO	3848	3848	3848	3848
StartClient	1555	1555	1555	1555

Versio 2.5.4 mallipohjaisten testien tulokset

Taulukko L3.7: Testimäärät.

Suoritetut testit		
Testin nimi	**Onnistuneet**	**Epäonnistuneet**
AnswerOldestCall	205 (69 %)	91 (31 %)
CloseIncidentForm	490 (96 %)	16 (4 %)
IVO - Add Units Condition	472 (100 %)	0
IVO - Add Units Probability	337 (100 %)	0
IVO - Answer Or Open Old Incident	341 (68 %)	158 (32 %)
IVO - Choose Unit Type	295 (100 %)	0
IVO - Dispatch Units Condition	506 (100 %)	0
IVO - Dispatch Units Probability	283 (100 %)	0
IVO - Incident Form	467 (92 %)	39 (8 %)
IVO - Login	1 (100 %)	0
IVO - Main Operation	349 (68 %)	158 (32 %)
IVO - Main Screen	341 (67 %)	165 (33 %)
IVO - Modelled tests	341 (67 %)	165 (33 %)
IVO - New Incident	294 (98 %)	3 (2 %)
IVO - Old Incident	150 (87 %)	21 (13 %)
IVO - Old Incident Probability	73 (77 %)	21 (23 %)
IVO - Risk Evaluation	482 (95 %)	24 (5 %)
IVO - Select Type Of Incident	231 (90 %)	24 (10 %)
SelectDispatchButton	227 (100 %)	0
SelectIncidentTypeAnimalRescue	70 (92 %)	6 (8 %)
SelectIncidentTypeBushFire	88 (90 %)	9 (10 %)
SelectIncidentTypeFalling	73 (89 %)	9 (11 %)
SelectIncidentTypeGroundless	107 (100 %)	0
SelectRandomIncident	144 (68 %)	67 (32 %)
SelectRoleIVO	1 (100 %)	0
SelectUnitsEmergency	72 (100 %)	0
SelectUnitsPolice	81 (100 %)	0
SelectUnitsRescue	73 (100 %)	0
SelectUnitsSocialServices	69 (100 %)	0
StartClient	1 (100 %)	0

Taulukko L3.8: Testien vasteajat.

Testien vasteajat				
Testin nimi	**Suurin**	**Pienin**	**Keskiarvo**	**Mediaani**
AnswerOldestCall	3375893	843	19478	4833
CloseIncidentForm	5528	108	447	238
IVO - Add Units Condition	11201	1	1226	1143
IVO - Add Units Probability	11182	1	1690	1540
IVO - Answer Or Open Old Incident	3375926	287	15558	4849
IVO - Choose Unit Type	11148	176	1888	1638
IVO - Dispatch Units Condition	11831	51	908	346
IVO - Dispatch Units Probability	518	1	231	316
IVO - Incident Form	41996	278	7197	4386
IVO - Login	4959	4959	4959	4959
IVO - Main Operation	3376128	498	15695	4961
IVO - Main Screen	3401935	1259	21901	10066
IVO - Modelled tests	3401975	1274	21933	10080
IVO - New Incident	39831	56	5481	2967
IVO - Old Incident Condition	39492	1	4135	19
IVO - Old Incident Probability	39471	1	7488	1933
IVO - Risk Evaluation	39853	13	4638	1398
IVO - Select Type Of Incident	39796	368	8168	3976

Testien vasteajat				
Testin nimi	**Suurin**	**Pienin**	**Keskiarvo**	**Mediaani**
SelectDispatchButton	518	52	285	331
SelectIncidentTypeAnimal-Rescue	35452	368	6483	2466
SelectIncidentTypeBushFire	39451	503	7317	2998
SelectIncidentTypeFalling	39795	812	10734	7892
SelectIncidentTypeGround-less	10695	51	2262	306
SelectRandomIncident	43262	268	9460	4023
SelectRoleIVO	3405	3405	3405	3405
SelectUnitsEmergency	8389	342	1878	1662
SelectUnitsPolice	11148	176	1595	1341
SelectUnitsRescue	6682	436	2120	1965
SelectUnitsSocialServices	10850	694	1994	1586
StartClient	1539	1539	1539	1539